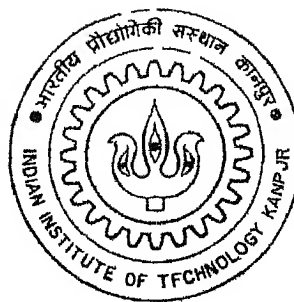


COMPARISON OF VARIOUS COMPENSATORY NEURON MODELS

BY

Murali Krishna Tallapudi



TH
EE/2000/M
T/44 c

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

AUGUST 2000

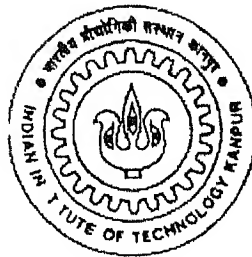
Comparison of Various Compensatory Neuron Models

A Thesis submitted
in partial fulfillment of the requirement
for the degree of

MASTER OF TECHNOLOGY

by

Murali Krishna Tallapudi



**DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY , KANPUR**

August 2000

के प्रा १७५
भा १ १७५

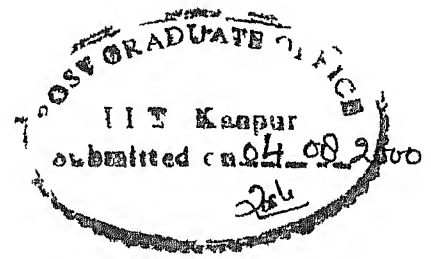
श्रीवाणि-१०१ १३३६१६

Li / 1000

11.

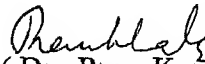


A133616



CERTIFICATE

This is to certify that the work contained in this thesis entitled **Comparison of Various Compensatory Neuron Models** " by Mr Murali Krishna Tallapudi has been carried out under my supervision and that has not been submitted else where for the degree


(Dr Prem Kumar Kalra)
Professor
Department of Electrical Engineering
Indian Institute of Technology Kanpur

ABSTRACT

The neuron model proposed by McCulloch & Pitts has a combination of aggregation and activation functions. This model requires a large no. of neurons in the standard neural network to solve any problem. To overcome this difficulty, compensatory neuron models have been proposed which form the basis of compensatory neural network architecture. A total of seven compensatory neuron models have been investigated in conjunction with selfscaling scaled conjugate gradient algorithm. The performance of one neuron model has been compared with the standard neural network with scaled conjugate gradient learning algorithm to show the efficacy of the compensatory model. These compensatory models are also compared and discussed in the work.

ACKNOWLEDGEMENT

I would like to express my gratitude to Dr P K Kalra my thesis supervisor for his inspiring guidance and sincere supervision throughout my work I would also like to thank him for the complete freedom afforded to me in my work as well as for the excellent facilities in the lab

My sincere thanks to M Sinha for his complete cooperation and moral support throughout my work I would like to thank mamana kiamu madhav piashnat sushil sieeram kalla krishnarao naidu ksudheer marim for their words of encouragement

Last but not least I am grateful to lord Krishna and my parents

Index

| | | |
|-------|---|----|
| 1 | Introductin | |
| 1 1 | An overview of Artificial Neural Networks | 1 |
| 1 2 | Existing neuron models | 2 |
| 1 2 1 | McCulloch Pitts model | 2 |
| 1 2 2 | McCulloch Pitts model with continuous Transfer function | 3 |
| 1 3 | The drawbacks existing neuron models ANN | 4 |
| 1 4 | Suggested Remedies | 4 |
| 1 5 | Organization of Thesis | 5 |
| 2 | | |
| 2 1 | Preliminary Remarks | 6 |
| 2 2 | Different Compensatory Neuron Models | 8 |
| 2 2 1 | Model 0 | 8 |
| 2 2 2 | Model 1 | 9 |
| 2 2 3 | Model 2 | 10 |
| 2 2 4 | Model 3 | 11 |
| 2 2 5 | Model 4 | 12 |
| 2 2 6 | Model 5 | 13 |
| 2 2 7 | Model 6 | 14 |
| 3 | Second Order Learning Algorithms | |
| 3 1 | Preliminary Remarks | 15 |
| 3 2 | Second Order Learning Algorithms | 16 |
| 3 2 1 | Scaled Conjugate Gradient Algorithm | 16 |
| 3 3 | Conjugate Gradient Algorithm | 17 |
| 3 3 1 | Self scaling scaled conjugate gradient algorithm | 20 |

| | | |
|---------|--------------------------|-----|
| 4 | Preliminary Remarks | |
| 4 1 | Functional Mapping | 21 |
| 4 1 1 | $\sin(x)\sin(y)$ problem | 21 |
| 4 2 | Classification | 22 |
| 4 2 1 | XOR Problem | 22 |
| 4 2 2 | Parity Problem | 24 |
| 4 2 3 | Half adder truth Problem | 28 |
| 4 2 4 | Spiral Problem | 28 |
| 5 | Results and Discussion | |
| 5 1 | Preliminary Remarks | 29 |
| 5 2 | Functional mapping | 29 |
| 5 2 1 | $\sin(x)*\sin(y)$ | 29 |
| 5 2 2 | Functional | 30 |
| 5 3 | Classification Problems | 31 |
| 5 3 1 | XOR Problem | 31 |
| 5 3 2 | Parity Problem | 32 |
| 5 3 2 1 | 4 bit Parity Problem | 33 |
| 5 3 2 2 | 5 bit Parity Problem | 33 |
| 5 3 2 3 | 6 bit Parity Problem | 33 |
| 5 3 2 4 | Half adder truth problem | 34 |
| 5 4 | Concluding Remarks | 34 |
| 6 | Closing comments | |
| 6 1 | Summary | 34a |
| 6 2 | Scope of future work | 34a |
| | References | 62 |

LIST OF FIGURES

- 1 1 General feedforward ANN architecture
- 1 2 Simple McCulloch & Pitts Model
- 1 3 McCulloch New Model
- 2 1 General compensatory neuron model
- 2 2 Schematic of compensatory Neural network architecture
- 4 1 $\sin(x) * \sin(y)$ Diagram
- 4 2 XOR problem
- 4 3 Spiral classification problem
- 5 1a A comparison of error convergence for the STD architectures and the compensatory model0 for $\sin(x)\sin(y)$ problem
- 5 1b convergence error for $\sin(x)\sin(y)$ problem on training set for compensatory models using SSCGA learning
- 5 2 Error convergence for the functional during the training
- 5 3a Comparison of Actual Vs Predicted values obtained using Model0 with SSCGA learning
- 5 3b Comparison of Actual Vs Predicted values obtained using Model1 with SSCGA learning
- 5 4a Comparison of Actual Vs Predicted values obtained using Model0 with SSCGA learning
- 5 4b Comparison of Actual Vs Predicted values obtained using Model1 with SSCGA learning
- 5 5 Error plot of XOR for CNNA with one Neuron
- 5 6 Error plot of XOR for CNNA with 2 Neurons
- 5 7 Error plot of XOR for CNNA with 3 Neurons
- 5 8 Error plot of XOR for CNNA with 4 Neurons
- 5 9 Error plot of 4 – bit parity problem for CNNA with one Neuron
- 5 10 Error plot of 4 – bit parity problem for CNNA with 2 Neurons
- 5 11 Error plot of 4 – bit parity problem for CNNA with 3 Neurons
- 5 12 Error plot of 4 – bit parity problem for CNNA with 4 Neurons
- 5 13 Error plot of 5 – bit parity problem for CNNA with one Neuron
- 5 14 Error plot of 5 – bit parity problem for CNNA with 2 Neurons
- 5 15 Error plot of 5 – bit parity problem for CNNA with 3 Neurons
- 5 16 Error plot of 5 – bit parity problem for CNNA with 4 Neurons
- 5 17 Error plot of 6 – bit parity problem for CNNA with one Neuron
- 5 18 Error plot of 6 – bit parity problem for CNNA

| | |
|------|--|
| | with 2 Neurons |
| 5 19 | Error plot of 6 – bit parity problem for CNNA with 3 Neurons |
| 5 20 | Error plot of 6 – bit parity problem for CNNA with 4 Neurons |
| 5 21 | Error plot of Half adder truth problem for CNNA With one neuron |
| 5 22 | Error plot of Half adder truth problem for CNNA With 2 neurons |
| 5 23 | Error plot of Half adder truth problem for CNNA With 3 neurons |
| 5 24 | Error plot of Half adder truth problem for CNNA With 4 neurons |

LIST OF TABLES

| | |
|-----|---|
| 4 1 | 4 Bit Parity Problem |
| 4 2 | 5 Bit Parity Problem |
| 4 3 | 6 Bit Parity Problem |
| 4 4 | Training set for Half adder truth problem |
| 5 1 | Results for the testing data set for model0 |
| 5 2 | Results for the testing data set for STD |

List of symbols

| | |
|-----------------|--|
| O_{Σ} | Output of Σ part of generalized neuron |
| W_{Σ} | Weight between Σ part of input side and Σ part of neuron on |
| $\tilde{w}(n)$ | Weights in the connection at the n^{th} iteration |
| δ_j | |
| a_h | h^{th} input to the network |
| a_i | i^{th} input to the network |
| ANN | Artificial neural network |
| BP | Backpropagation |
| CNNA | Compensatory neural network architecture |
| d_k | Desired output of the k^{th} output node |
| E | Error |
| netp_j | summation of inputs at j^{th} neuron neuron on output side |
| O_{Π} | Output of Π part of generalized neuron |
| O_k | k^{th} output |
| o_k | actual output of the k^{th} output node |
| O_{pk} | Output of generalized neuron output side |
| SCGA | Scaled conjugate gradient algorithm |
| SSCGA | Self scaled conjugate gradient algorithm |
| STD | Standard architecture |
| sum_j | Total input to j^{th} neuron |
| W_{Π} | Weight between Π part of neuron on input side and Σ part of |
| w_{pj} | weights between p^{th} neuron to j^{th} the neuron for Π type of neurons |
| w_{kj} | Weight between k^{th} output neuron to j^{th} compensatory neuron |
| x_j | Weighted sum of x_{pj} and x_{kj} |
| x_{p_j} | Output of the j^{th} activation function 1 |
| x_{p_j} | Output of the j^{th} activation function 2 |

CHAPTER 1

Introduction

1.1 An overview of Artificial Neural Networks

Over the last one decade Artificial neural networks (ANNs) have emerged as a new powerful tool for classification and functional mapping. As universal approximators, ANNs offer a systematic approach for these problems, especially the problems which are hard to analyze in full details. In general, neural networks can have four kinds of architecture [10], namely *single layer feedforward Networks*, *multilayer feedforward networks*, *recurrent neural networks*, and *lattice neural structure*. Among them, the most widely used architecture is feedforward neural networks, and the present work deals with them. A typical multilayer feedforward neural network is shown in fig 1.1. Multilayer feedforward neural networks have been applied successfully to solve some difficult and benchmark problems by training them.

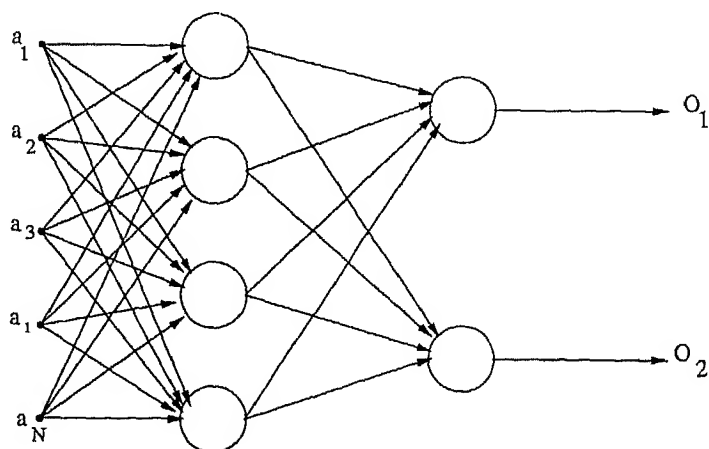


FIG 1.1 General Feedforward ANN architecture

1.2 Existing neuron models

Neural network models even neurobiological ones assume many simplifications over actual biological neural networks. Such simplifications are necessary to understand the intended properties and to attempt any mathematical analysis. In the following we discuss few models which are frequently used.

1.2.1 McCulloch –Pitts model

McCulloch and Pitts modeled simple logical units called cells so as to represent and analyze the logic of situations that arise in any discrete process. It is a simple two state machine [9]. Each cell is a finite state machine and accordingly operates in discrete time instants which is assumed to be synchronous among all cells. It contains an aggregation and a binary transfer function. When the sum of inputs exceeds some threshold value then the output of the neuron will be MAX otherwise MIN. Fig. 1.2 shows the basic structure of the McCulloch neuron model and the corresponding equation gives the state of the output. We can have unipolar and bipolar type of transfer functions.

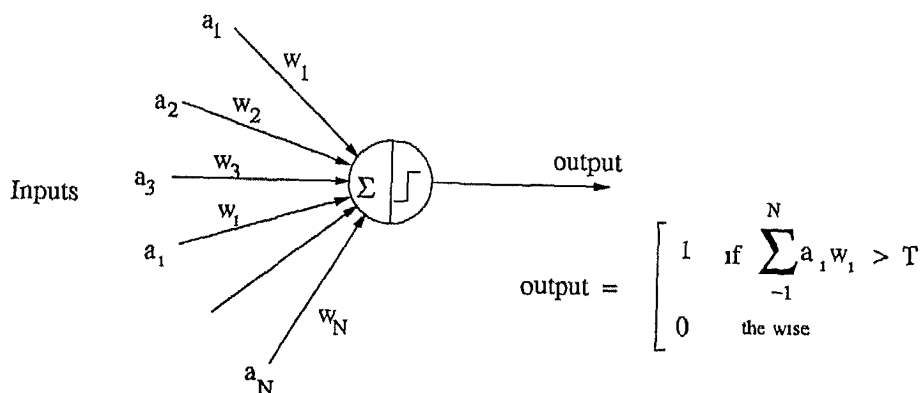


Fig 1.2 Simple McCulloch – Pitts Model

1.2.2 McCulloch – Pitts model with continuous Transfer function

The structure of this model looks like the previous model except the activation function. This will have a continuous activation function like linear, sigmoidal, gaussian, tan hyperbolic, etc. Fig 2.3 shows the diagram of this model.

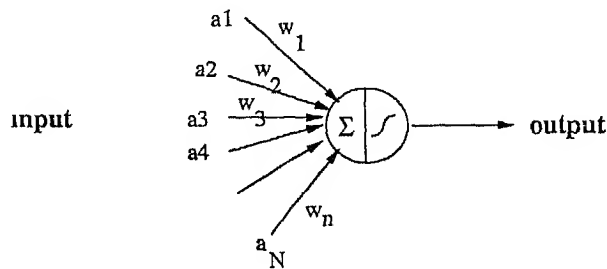


FIG 1.3 McCulloch New Model

The equations for some of transfer functions are

$$O_j = \sum_{i=1}^N W_{ji} a_i - T \quad \text{linear transfer characteristics}$$

$$O_j = \frac{1}{1 + \exp \left[-\lambda \left(\sum_{i=1}^N W_{ji} a_i - T \right) \right]} \quad \text{unipolar sigmoid transfer function}$$

$$O_j = \frac{2}{1 + \exp \left[-\lambda \left(\sum_{i=1}^N W_{ji} a_i - T \right) \right]} - 1 \quad \text{bipolar transfer function}$$

1.3 The drawbacks existing neuron models ANN

- 1 Number of neurons required in hidden layer are large for complex functions approximations
- 2 Number of hidden layers required for complicated function may be three or more. Even though it has been reported that a three layer network can approximate any functional relations, but the training time is too large for that.
- 3 Above mentioned constraints are not only computationally expensive in terms of convergence and large no of neurons in each layer, but also determining fault tolerant capabilities of the neural network. Several iterations are required before the ANN is trained to give accurate results.
- 4 Size of the ANN decides total no of unknowns to be determined and hence minimum no of training data (input – output pairs) required for development of neural network model. In case of complex functions the training data required is huge due to the requirement of large no of neurons and hidden layers.
- 5 The training time of the ANN depends on the input and output mappings like $X - Y$
 $\Delta Y - Y \propto \Delta Y \propto \Delta X \propto \Delta Y$

1.4 Suggested Remedies

The following suggestions have been made in the present work to alleviate the above mentioned shortcomings

- 1 Development of neuron models which are flexible to accommodate variations in its model and hence reduce the total number of neurons in the ANN.
- 2 These new neurons should also exhibit characteristics of existing neurons so that the models are general enough to accommodate the properties of the simple neurons to higher order neurons.
- 3) Total no of hidden layers required must also be reduced. This would result in the neural

network model which is computationally efficient

4) The new neuron models should not require more data for training i.e. reduction in free

parameters associated in the neurons

1.5 Organization of Thesis

Chapter 2 of this thesis contains the description of different types of new Neuron models of feed forward networks and respective derivations. Chapter 3 describes some of the second order learning algorithms. Chapter 4 will give you a brief idea about the bench mark problems which I considered. Chapter 5 presents the results for different models of neural network with different problems. Chapter 6 concludes the present work and giving aspects of further development of the present work.

Chapter 2

2.1 Preliminary Remarks

We have discussed the existing neuron models and their demerits in the previous chapter. It has been shown in literature that a three layer neural network can be a universal function approximator for a given input/output data. The existing neuron model has aggregation function with sigmoidal, radial basis, tangent hyperbolic or linear function as the activation function or nonlinearity. To overcome this deficiency, these conventional neuron models may not be computationally efficient to meet many real life applications. The proposed compensatory models have both sigmoidal and gaussian functions with weight sharing. The proposed new compensatory neurons have flexibility at both aggregation and threshold function level to cope with the nonlinearities. The proposed neuron has both Σ and Π aggregation functions. The final output of the neuron is function of two outputs O_Σ, O_Π and the weights w_Σ, w_Π respectively. Figure 2.1 shows the general structure of the proposed compensatory neuron model 1 and Figure 2.2 shows the complete architecture of CNNA with compensatory neurons in the hidden layer.

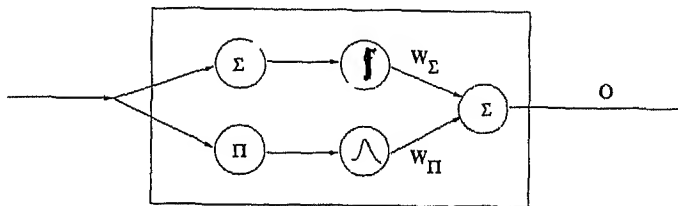


Fig 2.1 General compensatory Neuron Model

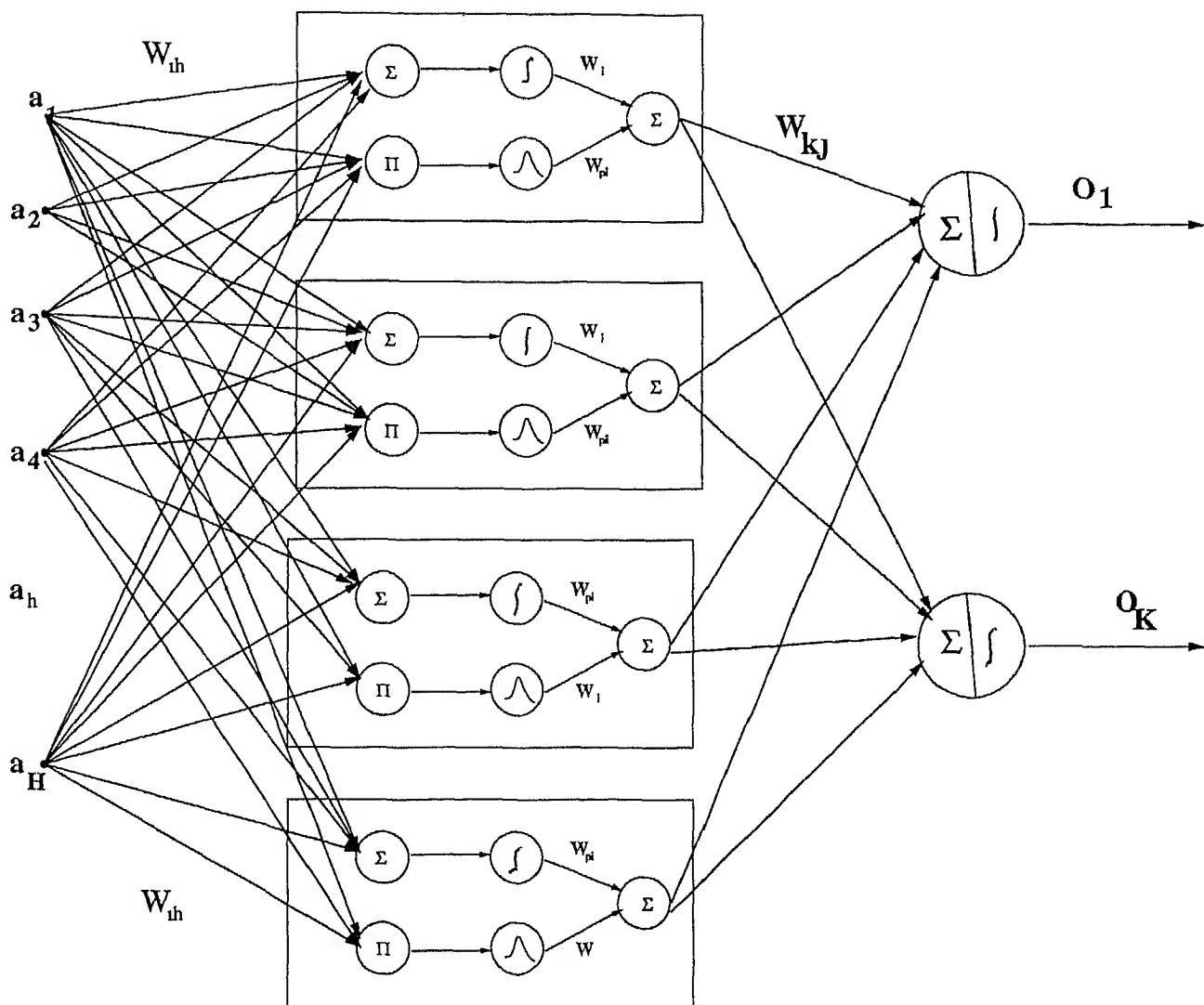


FIG 2 2 CNNA with 4 Hidden layer Neurons

2.2 Different Compensatory Neuron Models

2.2.1 MODEL 0

In this model both Σ as well as Π have been taken as the aggregation function and the output of these aggregation functions have been passed through the tan hyperbolic and the arctangent functions respectively. Finally the outputs of these activation functions are summed up to get the neuron output. The output of the compensatory neuron can be written as

$$x_j = xp_j w_{pj} + xp_j w_{pj} \quad 2.1$$

Described below are equations pertaining to weights update for different layers

Output layer weights update

$$\frac{\partial E}{\partial w_{kj}} = -(d_k - o_k)x_j = \delta_k x_j \quad 2.2$$

Neuron block weights update

$$\frac{\partial E}{\partial w_{pj}} = x_p \sum_{k=1}^K \delta_k w_{kj} \quad 2.3$$

$$\frac{\partial E}{\partial w_{pj}} = x_{pj} \sum_{k=1}^K \delta_k w_{kj} \quad 2.4$$

Input layer weights update

$$\frac{\partial E}{\partial w_{jh}} = w_{pj} \frac{\partial x_{pj}}{\partial net_{pj}} a_h \sum_{k=1}^K \delta_k w_{kj} \quad 2.5$$

$$\frac{\partial E}{\partial w_{jh}} = w_{pj} \frac{\partial x_{pj}}{\partial net_{pj}} a_h (sum_j - a_h w_{jh}) \delta_j a_h \quad 2.6$$

$$\text{where } sum_j = \sum_{h=1}^H a_h w_{jh}$$

2.2.2 Model 1

This model is similar to the model 0. The only difference is that in this model the weights associated with the output of the product aggregation function when passed through the arctangent function is $(1 - w_{\Sigma})$

Hence the output of the compensatory neuron can be written as

$$x_j = xp_j w_{pj} + xp_j (1 - w_{pj}) \quad 2.7$$

and the equations for the weights update are

Output layer weights update

$$\frac{\partial E}{\partial w_{kj}} = -(d_k - o_k) x_j = \delta_k x_j \quad 2.8$$

Neuron block weights update

Weights in the neuron blocks are compensatory

$$\frac{\partial E}{\partial w_{pj}} = -\frac{\partial E}{\partial w_{pj}} = \left(xp_j - xp_j \right) \sum_{k=1}^K \delta_k w_{kj} \quad 2.9$$

$$w_{pj} = 1 - w_{pj}$$

Input layer weights update

$$\frac{\partial E}{\partial w_{jh}} = w_{pj} \frac{\partial xp_j}{\partial net_{pj}} a_h \sum_{k=1}^K \delta_k w_{kj} \quad 2.10$$

$$\frac{\partial E}{\partial w_{jh}} = w_{pj} \frac{\partial xp_j}{\partial net_{pj}} a_h (sum_j - a_h w_{jh}) \delta_j a_h \quad 2.11$$

$$\text{where } sum_j = \sum_{h=1}^H a_h w_{jh}$$

Model 0 & Model 1 are known as summation type compensatory neuron models since the output of the tan hyperbolic and arctangent functions have been added up

2.2.3 Model 2

In Model – 2 outputs of the activation functions model are multiplied after being exponentiated to the power w_p and w_{Π}

Here the output of the neuron in the form of products is as given below

$$\begin{aligned}
 x_j &= [x_{pj}]^{w_{pj}} * [x_{pj}]^{w_{pj}} \\
 \text{or} \\
 x_j &= [x_{pj}]^{w_{pj}} * [x_{pj}]^{(1-w_{pj})}
 \end{aligned}
 \tag{2.12}$$

The output layer weight update is

$$\frac{\partial E}{\partial w_{kj}} = -(d_k - o_k) x_j = \delta_k x_j
 \tag{2.13}$$

Neuron block weights update is

$$\frac{\partial E}{\partial w_{pj}} = x_j \ln(x_{pj}) \sum_{k=1}^K \delta_k w_{kj}
 \tag{2.14}$$

$$\frac{\partial E}{\partial w_{pj}} = x_j \ln(x_{pj}) \sum_{k=1}^K \delta_k w_{kj}
 \tag{2.15}$$

and

$$-\frac{\partial E}{\partial w_{pj}} = \frac{\partial E}{\partial w_{pj}} = x_j \ln\left(\frac{x_{pj}}{x_{pj}}\right) \sum_{k=1}^K \delta_k w_{kj}
 \tag{2.16}$$

if the weights are compensatory i.e. $w_{pj} = 1 - w_{pj}$

The input layer weights update

$$\frac{\partial E}{\partial w_{jh}} = \frac{x_j}{x_{pj}} \left(\frac{\partial x_{pj}}{\partial net_{pj}} \right) a_h w_{pj} \sum_{k=1}^K w_{kj} \delta_k \quad 2.17$$

$$\frac{\partial E}{\partial w_{jh}} = \frac{x_j}{x_{pj}} \left(\frac{\partial x_{pj}}{\partial net_{pj}} \right) a_h w_{pj} \left(sum_j - a_h w_{jh} \right) \sum_{k=1}^K \delta_k w_{kj} \quad 2.18$$

2.2.4 Model 3

This neuron model has a complicated aggregation function which is neither summation function nor a product function alone but a combination of two

The output of this neuron model is

$$x_j = (x_{pj} + x_{pj} - x_{pj} x_{pj}) (1 - w_{pj}) + x_{pj} w_{pj} \quad 2.19$$

Output layer weights update

$$\frac{\partial E}{\partial w_{kj}} = -(d_k - o_k) x_j = \delta_k x_j \quad 2.20$$

Neuron Block weights update

$$\frac{\partial E}{\partial w_{pj}} = x_{pj} (x_{pj} - 1) \sum_{k=1}^K \delta_k w_{kj} \quad 2.21$$

Input layer weights update

$$\frac{\partial E}{\partial w_{jh}} = \frac{\partial x_{pj}}{\partial net_j} \left[(1 - w_{pj}) (1 - x_{pj}) + w_{pj} \right] a_h \sum_{k=1}^K \delta_k w_{kj} \quad 2.22$$

$$\frac{\partial E}{\partial w_{jh}} = \frac{\partial x_{pj}}{\partial net_j} \left[(1 - w_{pj}) (1 - x_{pj}) \right] a_h (sum_j - w_{jh} a_h) \sum_{k=1}^K \delta_k w_{kj} \quad 2.23$$

2.2.5 Model 4

This model is similar to the Model 3 but the output of the neuron is in the product form as follows

$$x_J = \left(x_{p_J} + x_{p_J} - x_{p_J} x_{p_J} \right)^{(1-w_{PJ})} x_{p_J}^{w_{PJ}} \quad 2.24$$

Output layer weights update

$$\frac{\partial E}{\partial w_{kJ}} = -(d_k - o_k) x_J = \delta_k x_J \quad 2.25$$

Neuron Block weight update

$$\frac{\partial E}{\partial w_{PJ}} = x_J \ln \left(\frac{x_{p_J}}{(x_{p_J} + x_{p_J} - x_{p_J} x_{p_J})} \right) \sum_{k=1}^K \delta_k w_{kJ} \quad 2.26$$

Input layer weights update is

$$\frac{\partial E}{\partial w_{jh}} = x_J \frac{\partial x_{p_J}}{\partial net_J} \left[\frac{(1-w_{PJ})(1-x_{p_J})}{(x_{p_J} + x_{p_J} - x_{p_J} x_{p_J})} + \frac{w_{PJ}}{x_{p_J}} \right] a_h \quad 2.27$$

$$\frac{\partial E}{\partial w_{jh}} = x_J \frac{\partial x_{p_J}}{\partial net_J} \left[\frac{(1-w_{PJ})(1-x_{p_J})}{(x_{p_J} + x_{p_J} - x_{p_J} x_{p_J})} \right] + a_h (sum_J - w_{jh} a_h) \quad 2.28$$

2.2.6 Model 5

This is also summation neuron model however it uses the arithmetic mean and geometric mean of the output of the activation functions as shown below

$$x_j = \left(\frac{x_{p_j} + xp_j}{2} \right) (1 - w_{pj}) + \sqrt{x_{p_j} xp_j w_{pj}} \quad 2.29$$

Output layer weight update is

$$\frac{\partial E}{\partial w_{kj}} = -(d_k - o_k) x_j = \delta_k x_j \quad 2.30$$

Neuron Block weights update is

$$\frac{\partial E}{\partial w_{pj}} = \frac{1}{2} \left(\sqrt{xp_j} - \sqrt{x_{p_j}} \right)^2 \sum_{k=1}^K \delta_k w_{kj} \quad 2.31$$

Input layer weights update is

$$\frac{\partial E}{\partial w_{jh}} = \frac{1}{2x_{p_j}} \frac{\partial x_{p_j}}{\partial net_j} \left[(1 - w_{pj}) x_{p_j} + \sqrt{x_{p_j} xp_j w_{pj}} \right] a_h \sum_{k=1}^K w_{kj} \delta_k \quad 2.32$$

$$\frac{\partial E}{\partial w_{jh}} = \frac{1}{2xp_j} \frac{\partial xp_j}{\partial net_j} \left(sum_j - a_h w_{jh} \right) \left[(1 - w_{pj}) x_{p_j} + \sqrt{x_{p_j} xp_j w_{pj}} \right] \quad 2.33$$

2.2.7 Model 6

This model is also similar to the Model 5 but the output is in the product form. The output is

$$x_j = \left(\frac{x_{pj} + xp_j}{2} \right)^{(1-w_{pj})} \sqrt{x_{pj} xp_j}^{w_{pj}} \quad 2.34$$

The output layer weights update is

$$\frac{\partial E}{\partial w_{kj}} = -(d_k - o_k)x_j = \delta_k x_j \quad 2.35$$

The neuron Block weights update is

$$\frac{\partial E}{\partial w_{pj}} = -x_j \ln \left(\frac{\left(\frac{x_{pj} + xp_j}{2} \right)^{1-w_{pj}} \sqrt{x_{pj} xp_j}^{w_{pj}}}{\sqrt{x_{pj} xp_j}} \right) \sum_{k=1}^K \delta_k w_{kj} \quad 2.36$$

The input layer weight update is

$$\frac{\partial E}{\partial w_{jh}} = a_h x_j \frac{\partial x_{pj}}{\partial net_j} \left[\frac{(1-w_{pj})}{\left(\frac{x_{pj} + xp_j}{2} \right)^{1-w_{pj}}} + \frac{w_{pj}}{2x_{pj}} \right] \sum_{k=1}^K w_{kj} \delta_k \quad 2.37$$

$$\frac{\partial E}{\partial w_{jh}} = a_h \frac{\partial xp_j}{\partial net_j} \left(sum_j - a_h w_{jh} \right) \left[\frac{(1-w_{pj})}{\left(\frac{x_{pj} + xp_j}{2} \right)^{1-w_{pj}}} + \frac{w_{pj}}{2x_{pj}} \right] \sum_{k=1}^K w_{kj} \delta_k \quad 2.38$$

CHAPTER 3

SECOND ORDER LEARNING ALGORITHMS

3.1 Preliminary Remarks

The first order learning algorithms have limitations such as divergence, slow convergence and degree of accuracy achieved is generally lower. This may be attributed to the gradient descent method which is only an approximation of truncated Taylor series. However, another reason for the slow convergence of BP training is the occurrence of the phenomenon of premature saturation of ANN output units which is applicable to both the first and second order and second order learning when the units are mapped by sigmoid like functions [2]. This is characterized by the temporary trapping of the ANN output units at saturated activation levels during the early stage of training process. While thus trapped, the saturated output units preclude any significant improvements in the training weights directly connected to these units, causing unnecessary increase in the number of iterations required to train the ANN.

There may be cases in which learning speed of first order STD BP is a limiting factor in practical application of ANN to problems that require high accuracy in the ANN mapping function. The problems that belong to these classes are related to system identification.

A nonlinear modeling, time series prediction, navigation, manipulation and robotics. In addition, the STD BP [3] requires selection of appropriate parameters by the user that is mainly carried out by a trial and error process. Since one of the competitive advantages of the neural networks is the ease with which they may be applied to novel or poorly understood problems, it is imperative to consider automated and robust learning methods with a good average performance on many classes of problems. Many *ad hoc* schemes have been suggested for improving the capability of first order learning schemes but some of them are problem specific while others simply increase the computational complexity while the improvement in

learning may not be significant [4] To overcome this limitation second order learning schemes have been suggested that have been shown to accelerate the convergence of the learning phase on a variety of problems The divergence of these algorithms is not completely ruled out as often the *Hessian* matrices become non positive definite In this chapter second order learning algorithms such as scaled conjugate gradient algorithm (SCGA) and self scaling scaled conjugate gradient (SSCGA) are described which tries to overcome some of these limitations

3.2 Second Order Learning Algorithms

The greatest difficulty in using the steepest descent gradient algorithm is that a one dimensional minimization in direction **a** followed by a minimization in direction **b** does not imply that the function is minimized on the sub space generated by **a** and **b** Minimization along direction **b** in general spoils a previous minimization along direction **a** On the contrary if the directions were non interfering and linearly independent at the end of n steps the process would converge to the minimum of the quadratic function

$$Q(\tilde{w}) = c^T \tilde{w} + 0.5 \tilde{w}^T G \tilde{w}$$

where G is symmetric and positive definite and N is the dimension of the weight space

The concept of non interfering directions is the basis of conjugate gradient algorithm (CGA) for minimization But this method requires a line search scheme to find the step size to descend in a conjugate direction Unfortunately this line search may not work if the function being optimized is non quadratic as is the case frequently encountered in the ANN In the following sections improved models of the CGA are described which avoid the *Hessian* calculation but are as good as any second order algorithm based on *Hessian* calculation

3.2.1 Scaled Conjugate Gradient Algorithm

SCGA which is an improvement over the conjugate gradient algorithm (CGA) was developed by Moré in 1993 based on the Levenberg Marquardt approach in order to scale the step size [5] CGA is based on the fact that minimization of a positive definite

quadratic function is equivalent to solving a system of linear equations obtained on setting gradient to zero. A detailed discussion of CGA can be found in the references [6]. In CGA the step size is usually determined by one of the line search methods. But in spite of such a sound theoretical backing it often fails and converges to non stationary points. This may be attributed to the *Hessian* matrices becoming non positive definite. The quadratic approximation which holds good only in the neighborhood of the current point may not hold for points outside [7]. To overcome this limitation Moller proposed the SCGA by combining model trust region approach with the CGA [5]. While formulating the ANN he retained non linearity in the output. Here we remove the non linearity from the output and keep only the aggregation function as the output neurons and combine it with the SCGA learning and refer to it as SSCGA. This may facilitate the scaling of the output in the higher ranges. Besides giving higher accuracy it may also reduce significantly the number of iterations and hence computation. Moreover it rules out the possibility premature saturation as can be seen in the orbit determination section where it is able to optimize with the outputs with out normalization as well as when normalized in the higher range where certainly the STD with steepest descent LG CGA or SCGA will not work.

3.3 Conjugate Gradient Algorithm

Conjugate direction method from which conjugate gradient method is derived is thoroughly described in the reference [6]. Here an outline of the CGA is described which is the basis for SCGA and the SSCGA. Conjugate direction methods which are based on the pseudo optimization strategy described in the last chapter. Choose the search direction and the step size more carefully by using information from the second order approximation given by

$$E(\tilde{w} + \tilde{y}) \approx E(\tilde{w}) + E(\tilde{w})^T \tilde{y} + 0.5 \tilde{y}^T E(\tilde{w}) \tilde{y}$$

We quote here two theorems to summarize the working of the conjugate gradient method

Theorem # 1 let \tilde{p}_1, \tilde{p}_N be a conjugate system and \tilde{y}_1 a point in the white space. Let points $\tilde{y}, \tilde{y}_{N+1}$ be recursively defined by $\tilde{y}_{k+1} = \tilde{y}_k + \alpha_k \tilde{p}_k$ where $\alpha_k = \frac{\mu_k}{\delta_k}$, $\mu_k = -\tilde{p}_k^T E_q(\tilde{y}_k)$, $\delta_k = -\tilde{p}_k^T E_q(\tilde{y}_{k+1})$. Then \tilde{y}_{k+1} minimizes E_q restricted to the k plane Π_k given by \tilde{y}_1 and $\tilde{p}_1, \dots, \tilde{p}_k$ where $E(\tilde{y}) \approx E(\tilde{y}_1) + E(\tilde{y}_1)^T \tilde{y} + 0.5 \tilde{y}^T E(\tilde{y}_1) \tilde{y}$.

The conjugate direction theorem#1 assumes that a conjugate system is given. But this is not necessary and the conjugate vectors $\tilde{p}_1, \dots, \tilde{p}_N$ can be determined recursively. This is achieved by setting \tilde{p}_1 to this steepest descent vector $-E_q(\tilde{y}_1)$. Then \tilde{p}_{k+1} is determined recursively as a linear combination of the current descent vector $-E_q(\tilde{y}_{k+1})$ and the previous direction \tilde{p}_k . Theorem#2 concludes the above statement in short.

Theorem # 2 Let \tilde{y}_1 be a point in the weight space and \tilde{p}_1 and \tilde{r}_1 equal to the steepest descent vector $-E_q(\tilde{y}_1)$. Define \tilde{p}_{k+1} recursively by $\tilde{p}_{k+1} = \tilde{r}_k + \beta_k \tilde{p}_k$ where $\tilde{r}_{k+1} = E_q(\tilde{y}_{k+1})$, $\beta_k = \frac{(\tilde{r}_{k+1}^T \tilde{r}_k)}{\tilde{p}_k^T \tilde{r}_k}$ and \tilde{y}_{k+1} is the point generated in theorem#1. Then \tilde{p}_{k+1} is the steepest descent vector to E_q restricted to the $(N-k)$ plane Π_{N-k} conjugate to Π_k given by \tilde{y}_1 and $\tilde{p}_1, \dots, \tilde{p}_k$.

The vectors defined by theorem are referred to as conjugate direction and theorems 1 and 2 combined together give the CGA. Error will converge in just N steps if the error function is strictly a quadratic function.

Scaled conjugate gradient algorithm The line search technique that CGA uses for determining the optimum step size is a costly computational affair. Needless to say, CGA may fail in general and may converge to non stationary point because the algorithm works only with positive definite Hessian matrices and the quadratic approximations on which it is based may not hold when the current point is far from the desired minimum. This problem is tackled by combining the model test region approach known as Levenberg Marquardt algorithm with the CGA. The algorithm for updating the weights using the scaled conjugate gradient algorithm (SCGA) is given below for convenience [50]. The algorithm is as follows:

Step 1 Choose weight vector \tilde{w}_1 and scalars $0 < \sigma \leq 10^{-4}$ $0 < \lambda_1 \leq 10^{-6}$ $\lambda_1 = 0$ Set $\tilde{p}_1 = \tilde{r}_1 = -L(\tilde{w}_1)$ $k = 1$ and success=true

Step 2 If success=true then calculate second order information $\sigma_k = \sigma / |\tilde{p}_k|$ $\tilde{s}_k = (E(\tilde{w}_k + \sigma_k \tilde{p}_k) - E(\tilde{w}_k)) / \sigma_k$ $\delta_k = \tilde{p}_k^T \tilde{s}_k$

Step 3 Scale δ_k $\delta_k = \delta_k + (\lambda_k - \bar{\lambda}_k) |\tilde{p}_k|^2$

Step 4 If $\delta_k \leq 0$ then make the Hessian matrix positive definite $\bar{\lambda}_k = 2 \left(\bar{\lambda}_k - \delta_k / |\tilde{p}_k|^2 \right)$ $\delta_k = -\delta_k + \lambda_k |\tilde{p}_k|^2$ $\lambda_k = \bar{\lambda}_k$

Step 5 Calculate the step size $\mu_k = \tilde{p}_k^T \tilde{r}_k$ $\alpha_k = \mu_k / \delta_k$

Step 6 Calculate the comparison parameter $\Delta_k = 2\delta_k [E(\tilde{w}_k) - E(\tilde{w}_k + \alpha_k \tilde{p}_k)] / \mu_k$

Step 7 If $\Delta_k \geq 0$ then a successful reduction in error can be made

$$\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k \tilde{p}_k$$

$$\tilde{r}_{k+1} = -L(\tilde{w}_{k+1})$$

$$\tilde{\lambda}_k = 0 \text{ success} = \text{true}$$

If $k \bmod N = 0$ then restart algorithm

$$\tilde{p}_{k+1} = \tilde{r}_{k+1}$$

else

$$\beta_k = \left(|\tilde{r}_{k+1}|^2 - \tilde{r}_{k+1}^T \tilde{r}_k \right) / \mu_k$$

$$\tilde{p}_k = \tilde{r}_{k+1} + \beta_k \tilde{p}_k$$

If $\Delta_k \geq 0.75$ then reduce the scale parameter

$$\lambda_k = 0.25 \lambda_k$$

else

$$\bar{\lambda}_j = \lambda_k$$

success = false

Step 8 If $\Delta_k < 0.25$ then increase the scale parameter $\lambda_k = \lambda_k + \left(\delta_k (1 - \Delta_k) / |\bar{p}_k|^{-1} \right)$

Step 9 If the steepest descent direction $\tilde{r}_k \neq \tilde{0}$ then set $k = k + 1$ and go to 2 else terminate and return \tilde{w}_{k+1} as the desired minimum

3.3.1 Self scaling scaled conjugate gradient algorithm

When the non linearity is removed from the output layer the weights therein can be updated using a linear scheme. In the linear schemes singular value decomposition can be applied to solve the output layer weights and then rest of the weights are updated using the usual BP schemes either first order or the second order. Here we choose to update all the weights using the BP with SCGA algorithm and term it as self scaling scaled conjugate algorithm (SSCGA). Updating the weights in this manner avoids any premature saturation of the ANN moreover gives a better generalization which the mixed scheme can't do without regularization. Moreover the outputs may be scaled in the higher range for the purpose of training which is not possible by first order learning algorithm.

Chapter 4

Preliminary Remarks

There are some problems used by researchers as benchmark problems for testing the efficacy of ANN models and the learning algorithms developed. For example XOR or PARITY problems are considered as hard and static mapping/classification problems. In this work we test the compensatory ANN models in conjunction with self scaling scaled Conjugate Gradient algorithm and classify them in two categories 1) Functional mapping 2) Classification. In functional mapping we consider $\sin(x)\sin(y)$ and another functional mapping problem defined later in this chapter. In classification category we considered XOR, Parity, Spiral and Half adder truth problems. In parity problem again we considered four bit, five bit and six bit parity problems.

FUNCTIONAL MAPPING

1 $\sin(x)\sin(y)$ problem

$\sin(x)\sin(y)$ is a general functional mapping problem which is used by researchers to test the network capabilities. The equation which describes the function is

$$z(x, y) = \sin(x)\sin(y) \quad 4.1$$

this function gets more complex when the norm of the input vector (x, y) grows. We have generated a training set containing 2500 training patterns by varying the values of x and y in the range $(0, 5\pi)$. Fig 4.1 shows the $\sin(x)\sin(y)$ mapping.

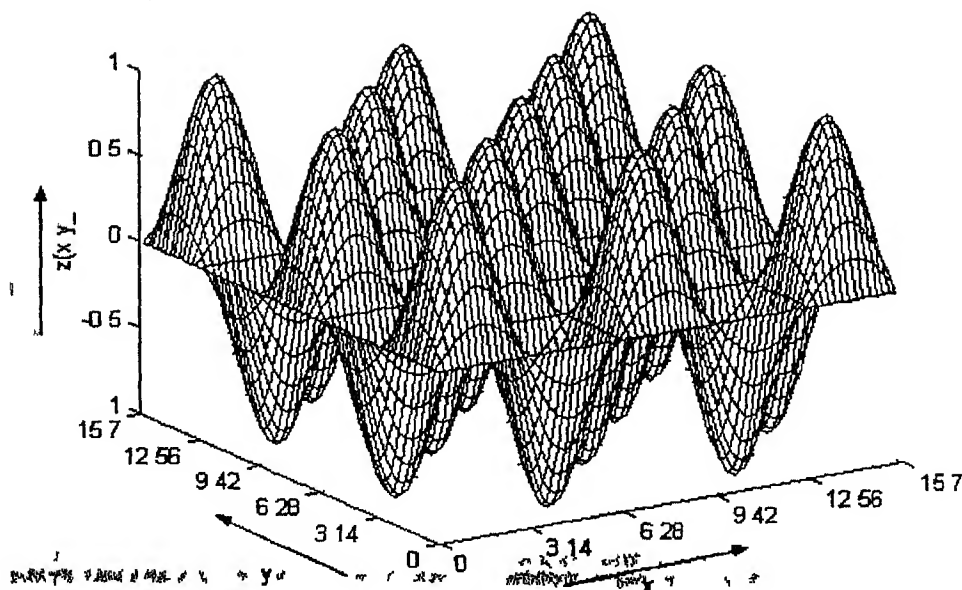


FIG 4 1 $\sin(x) * \sin(y)$ Diagram

2 Functional

$$z = \exp(-(x + y)) + x^2 y + 10/(y + x^2) + \exp(-\sin(x) + \cos(y)) + (x + y)^2$$

CLASSIFICATION

XOR Problem

The Exclusive – OR problem is the classic problem requiring hidden units. The XOR problem as compared with other logic operations (AND, OR, and their negates NAND, NOR) is nonlinearly separable. The training set for XOR problem is

$$\{(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0)\}$$

The training set is precise representing a set where no measurement errors /noise occurs. On several occasions the system gets trapped in a local minimum depending on the initial parameters of ANN. This is true for first order and second order learning algorithms. The new models are trained for the XOR problem and are also tested on a testing set to validate its efficacy.

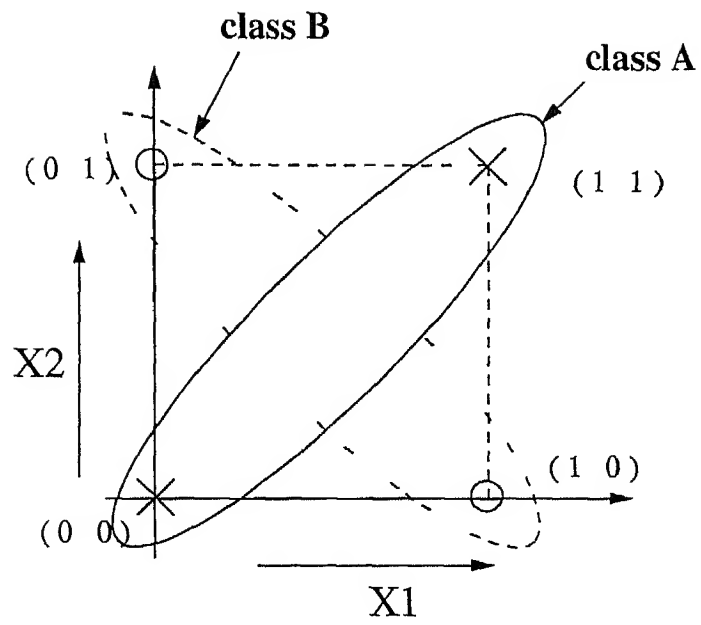


Fig 4 2 XOR Problem

Parity Problem

The N input parity problem has been a popular benchmark problem among researchers in ANN. This problem consists of mapping an N bit wide binary number into its parity i.e. if the input pattern consists of odd no. of ones then the parity is one else it is 0. This is considered as the difficult problem because the patterns that are closer (using the Euclidean distance) in the sample space i.e. numbers that differ in only one bit require their answers to be different. The XOR is a 2 input parity problem and the general solution for the parity problem is a group of XOR circuits. This benchmark is considered as a perfect training set. The training data for 4 bit parity problem is shown in table 4.1. The training data for 5 bit parity problem is given in Table 4.2. The training data for 6 bit parity problem is given in Table 4.3.

TABLE 4.1 4 – Bit Parity Problem

| X1 | X2 | X3 | X4 | Y |
|----|----|----|----|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

TABLE 4 2 5 – BIT PARITY PROBLEM

| X1 | X2 | X3 | X4 | X5 | Y |
|----|----|----|----|----|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

TABLE 4 3 6 – BIT PARITY PROBLEM

| X1 | X2 | X3 | X4 | X5 | X5 | Y |
|----|----|----|----|----|----|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |

HALF ADDER TRUTH PROBLEM

This is a relatively simple problem compare to parity problem where there are two inputs and two outputs This problem is a combination of two input XOR & AND problems This should classify both XOR & AND Table 4 4 shows the training set

| X1 | X2 | Y1 | Y2 |
|----|----|----|----|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

SPIRAL PROBLEM

Fig 4 3 shows the spiral classification problem

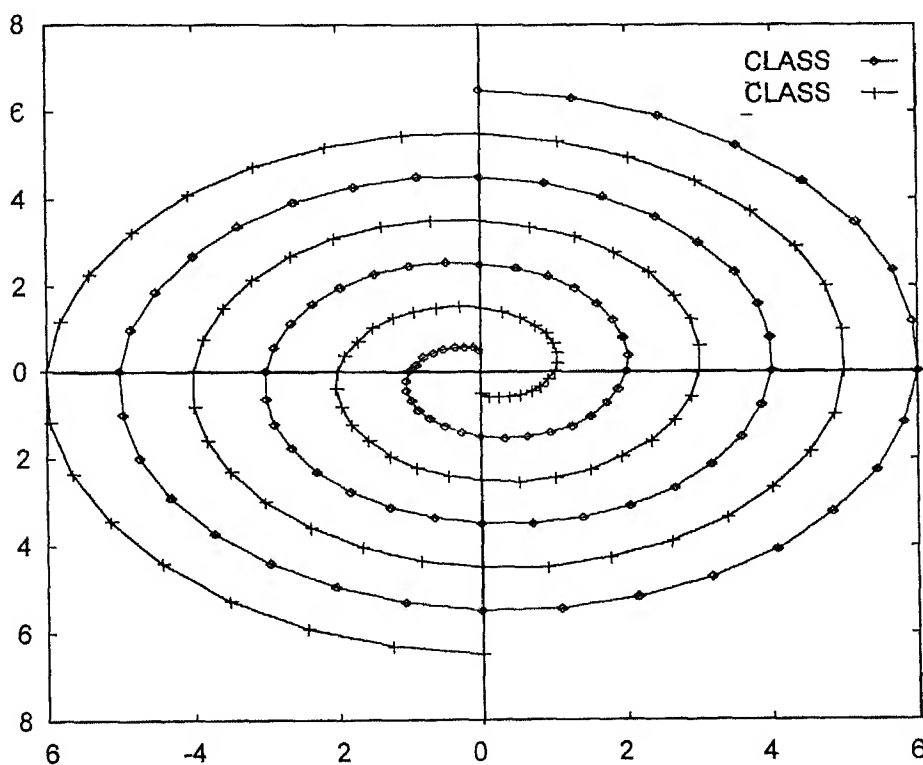


FIG 4 3 SPIRAL CLASSIFICATION PROBLEM

Chapter 5

Results and Discussion

5.1 Preliminary Remarks

In chapter 4 the various benchmark problems were discussed pertaining to functional mapping and the classification. Any new neuron model developed must be tested on these benchmark problems to validate its efficacy. Here in this chapter we test the compensatory models discussed earlier in chapter 2 on the benchmark problems described in chapter 4.

5.2 Functional mapping

This may involve mapping from a lower dimension to a higher dimensional system or vice versa. Essentially the capability of mapping a function depends upon the neuron model and the architecture used. In the following section we first test on $\sin(x)*\sin(y)$ problem. Throughout the text the error on original scale with number of iterations is presented.

5.2.1 $\sin(x)*\sin(y)$

Here the solution to this problem has been attempted using compensatory models with the SSCGA learning. A comparative study of the STD with the SCGA learning and compensatory model0 with the SSCGA learning is also presented. In Fig. 5.1a the convergence results for the STD with the SCGA and model0 with the SSCGA learning is shown. Here the std 6 6 1 refers to the standard architecture with 6 neurons in input layer and 6 in the hidden layer and 1 in the output layer. Here for the compensatory the model0 6 neuron blocks have been taken. The number of activation functions and the weights involved in the std 6 6 1 and the model0 are the same. But it is obvious that the convergence in the case of STD is very slow with this architecture. Increasing the number of neurons in the hidden layer in the case of STD to 6 15 1 has improved the result.

considerably but the number of neurons and the weights involved in this case has got doubled as compared to the model0

This is expected as the compensatory model is based on higher order neuron models as discussed in the earlier chapters. An early and better convergence can be achieved using the compensatory models. This may be advantageous for some critical industrial processes where the decision has to be taken fast.

In Fig 5 1b the convergence behavior of various compensatory models are presented. Here it can be observed that the performance of model6 is poor as compared to other models. Also the performance of model2 and model4 are not as good as the performance of model0, model1, model3 and model5. This may be attributed to the fact that the power and the multiplication terms provide a bias towards higher order terms while the first order terms get less weightage and therefore the performance of models with power terms multiplied become relatively sluggish. Performance of model0, model1, model3 and model5 are comparable. In the case of model0 and model1 the initial convergence is faster. The plots of actual vs predicted value for the training set is not presented for the sake of convenience as this periodic function produces the same output over different periods and therefore their values overlap on the graph making it blurred.

5 2 1 Functional

The equation of the functional is described in chapter 4. Here the data for training is generated by varying the values of x and y by 0.25 and 0.5 respectively over hundred intervals. The training set consists of 83 data points while the testing set consists of 17 data points. The results of training and testing have been presented in Figs 5 2-5 4. In Fig 5 2 the error convergence during training has been plotted against the number of iterations for the various compensatory models. Again it is obvious that the performance of model6 is poor as compared to other models. This reinforces the results obtained for the $\sin(x)*\sin(y)$ problem. Here model2 appears to work better as compared to other models. This may be attributed to the initialization of the ANN parameters as in the latter section it may be seen that model2, model4 and model6 perform little sluggish as compared to the other compensatory models. The performance of compensatory models with power terms need further investigation with the initialization of network parameters.

which may be set for the future work. The performances of other models are comparable. In Fig. 5.3 predicted values against the actual values of the output have been plotted for the training set for the model0 and model1. The observed error in training is less than 0.004% for individual outputs. This shows that network is able to learn the functional hidden in the data. In Fig. 5.4 predicted values against the actual values of the output have been plotted for the testing sets for model0 and model1. Here also in the prediction results error is less than 0.005% for individual entry. Results for the other models are not presented here as they are of the same nature as for the model0 and model1. Even in the case of model6 which does not converge properly the error is relatively significant at the lower values of the output.

5.3 Classification Problems

Here we test the compensatory neuron models on XOR, parity and half adder truth problems to validate their efficacy.

5.3.1 XOR Problem

The XOR problem is the simplest classification problem and requires a multi layer neural network to solve it. In fact using standard backpropagation algorithm with the STD it may take around 1000 iterations to classify this set properly. The second order learning algorithms make the convergence fast. It is found that it takes around 50 iterations to converge for the STD with 2-1-1 architecture with the SCGA learning. Here in Fig. 5.5 the error convergence for different compensatory models have been presented. It may be observed that the convergence of the compensatory models involving power terms is poor as compared to others. The convergence of the model0 and the model1 is faster. In the case of model0 and model1 the convergence is achieved just in 5-6 iterations as it is evident from the Fig. 5.5. In Fig. 5.6 and Fig. 5.7 increasing the number of neuron blocks from 1 to 2 and 3 respectively seem to have adverse effect on the convergence of model0 and model1. The performance of model2, model4 and model6 has improved drastically by increasing the number of neuron blocks. Again in Fig. 5.8 the convergence of model0 and model1 seem to improve. These fluctuations may be attributed to the initialization of the ANN parameters. The performance of model5 in Figs. 5.6-5.8 is consistently good.

and the convergence is fast too. In Tables 5.1 and 5.2 prediction results for the test data, for model0 with SSCGA learning and STD with SCGA learning respectively are presented. Here in STD 2-2-1 structure is taken while for model0 only 1 neuron block is taken. It is evident that the model0 with approximately half the number of neurons as compared to that for the STD gives much better generalization. However, it has been observed that increasing the number of neurons in hidden layer of the STD improves the result on testing set. Thus the performance of compensatory model appears to be superior to that of STD.

Table 5.1 Results for the testing data set for model0

| Input | | Output | Predicted output | Error |
|-------|-----|--------|------------------|-------|
| 0.1 | 0.9 | 1 | 0.91 | 0.09 |
| 0.8 | 0.2 | 1 | 0.78 | 0.22 |
| 0.3 | 0.2 | 0 | 0.30 | 0.30 |
| 0.8 | 0.8 | 0 | 0.22 | 0.22 |

Table 5.2 Results for the testing data set for STD

| Input | | Output | Predicted output | Error |
|-------|-----|--------|------------------|-------|
| 0.1 | 0.9 | 1 | 1.04 | 0.04 |
| 0.8 | 0.2 | 1 | 1.02 | 0.02 |
| 0.3 | 0.2 | 0 | 1.05 | 1.05 |
| 0.8 | 0.8 | 0 | 0.01 | 0.01 |

5.3.2 Parity Problem

The N parity problem has been used to compare different learning algorithms and comparing the epochs necessary to produce a perfect result (no misclassification). This problem will be explored here to test the different compensatory neuron models in conjunction with SSCGA learning using the same number of epochs of learning.

5 3 2 1 4 Bit Parity Problem

Figures 5 9 5 12 show the error convergence for the 4 bit parity problem In Fig 5 9 the error convergence for the different models using 1 neuron block is presented Here it is observed that none of the models converge Increasing the number of neuron blocks to 2 improves the convergence of all the models The initial convergence of model0 is faster as compared to other models The convergence of model3 and model5 seem to be better as compared to others In fact in the case of model0 and model1 the final slow convergence reflect that these are trapped in local minima In spite of this trapping in local minimum, model0 has classified in just 40 epochs

5 3 2 2 5 Bit Parity problem

The results for 5 bit parity problem are presented in Figs 5 13-5 16 From Fig 5 13 it is obvious that none of the models converges with one neuron block This is in accordance with the complexity of the problem As the complexity increases the number of neurons required also increases Therefore convergence for the one-neuron block is not expected In Fig 5 14 the convergence for different models are shown The plot for model0 is not included as its behavior is almost the same as that of model1 Also for the model6 convergence in the case of 5 bit parity problem is not observed with 1 2 3 and 4 neuron blocks therefore it is excluded from the plots for the 5 bit parity problem However it may be noted that increasing the number of neuron blocks improves the classifying power of the model6 The model3 and the model5 converge faster as compared to other models Convergence in the case of model1 is poor as compared to other models The performance of model2 improves with the increasing number of neuron blocks

5 3 2 3 6 Bit parity problem

In Figs 5 17 5 20 the plots for the 6 bit parity problem are presented It is obvious from all these figures that the model6 fail to converge It is seen that as the complexity of the classification problem is increasing it becomes more difficult for the model6 to classify it Also the performance of the model4 and the model2 is poor The performance of model3 and model5 is the best Here model1 shows some improvement with the increasing number of neurons

TABLE 5 3 RESULTS SUMMARY SHEET FOR FUNCTIONAL MAPPING WITH 6 NEURONS
IN THE HIDDEN LAYER

| | STD | Model0 | Model1 | Model2 | Model3 | Model4 | Model5 | Model6 |
|---------------|------|--------|--------|-----------|--------|---------|--------|--------|
| Sim(x)*Sim(y) | POOR | GOOD | GOOD | AVERAGE | GOOD | AVERAGE | GOOD | POOR |
| Function * | -- | GOOD | GOOD | VERY GOOD | GOOD | GOOD | GOOD | POOR |

$$\text{Function} = e^{(x+y)} + x \cdot y + \frac{1}{y+x} + e^{-(x+y)} + (x+y)$$

= not tested and hence results not available

TABLE 5 4 SUMMARY SHEET FOR CLASSIFICATION PROBLEM

| | No of Neurons In the Hidden layer | X OR | PARITY | | | HALF ADDED TRUTH |
|---------|--------------------------------------|------|---------|---------|---------|---------------------|
| | | | 4 - BIT | 5 - BIT | 6 BIT | |
| MODEL 0 | 1 | GOOD | POOR | | | POOR |
| | 2 | GOOD | AVERAGE | | | GOOD |
| | 3 | GOOD | | | | GOOD |
| | 4 | GOOD | | | | GOOD |
| MODEL 1 | 1 | GOOD | POOR | POOR | POOR | POOR |
| | 2 | GOOD | AVERAGE | POOR | AVERAGE | GOOD |
| | 3 | GOOD | GOOD | POOR | AVERAGE | GOOD |
| | 4 | GOOD | GOOD | POOR | AVERAGE | GOOD |
| MODEL 2 | 1 | POOR | POOR | POOR | POOR | POOR |
| | 2 | GOOD | GOOD | POOR | POOR | POOR |
| | 3 | GOOD | GOOD | POOR | POOR | GOOD |
| | 4 | GOOD | GOOD | POOR | POOR | GOOD |
| MODEL 3 | 1 | GOOD | POOR | POOR | POOR | POOR |
| | 2 | GOOD | AVERAGE | POOR | AVERAGE | GOOD |
| | 3 | GOOD | GOOD | AVERAGE | AVERAGE | GOOD |
| | 4 | GOOD | GOOD | AVERAGE | GOOD | GOOD |

TABLE 55 SUMMARY SHEET FOR CLASSIFICATION PROBLEM

| | No of Neurons In the Hidden layer | X OR | PARITY | | | HALF ADDED TRUTH |
|---------|--------------------------------------|------|---------|---------|---------|---------------------|
| | | | 4 BIT | 5 BIT | 6 BIT | |
| MODEL 4 | 1 | POOR | POOR | POOR | POOR | POOR |
| | 2 | GOOD | AVERAGE | POOR | POOR | POOR |
| | 3 | GOOD | AVERAGE | POOR | POOR | GOOD |
| | 4 | GOOD | AVERAGE | POOR | POOR | GOOD |
| MODEL 5 | 1 | GOOD | POOR | POOR | POOR | POOR |
| | 2 | GOOD | AVERAGE | POOR | AVERAGE | POOR |
| | 3 | GOOD | GOOD | AVERAGE | AVERAGE | GOOD |
| | 4 | GOOD | GOOD | GOOD | GOOD | GOOD |
| MODEL 6 | 1 | POOR | POOR | POOR | POOR | POOR |
| | 2 | GOOD | POOR | | POOR | POOR |
| | 3 | GOOD | AVERAGE | | POOR | AVERAGE |
| | 4 | GOOD | AVERAGE | | POOR | GOOD |

= FOR THOSE CASES TRAINING WAS NOT DONE

Chapter 6

Closing Comments

6.1 Summary

In this work seven compensatory models have been trained in conjunction with self scaling scaled conjugate gradient algorithm (SSCGA). These models are tested on benchmarking problems. The results for model0 with SSCGA are also compared with that of STD with scaled conjugate gradient algorithm (SCGA). It is found that the performance of the compensatory model is better than that of the STD. Moreover, a large computational saving is achieved using the compensatory model. Among the compensatory models, the model3 and model5 perform better than other models. The models involving the power terms of the output of the activation functions and their multiplication have sluggish performance. The compensatory models do not work on spiral problems.

6.2 Scope for future work

Needless to say, it may be worthwhile to point out that there is considerable scope for future work. Here, we summarize some of the summations for achieving the objective as indicated below:

- ✦ The power models can be further explored for initialization of the ANN parameters.
- ❖ The spiral problem may be attempted by introducing one more layer of compensatory neurons.
- For all the compensatory models, the net to the activation function can be altered by summing up the multiplication and the summation term before feeding it to the activation.
- One of the activation functions of the neuron block can be dropped and the compensatory summation of the summation and multiplication of weighted inputs may be taken.

From the results of 4 bit 5 bit and 6 bit parity problems it may be concluded that the performance of the summation models are better than that of the power models. Among summation models, model3 and model5 converge better than the model1 for complex problems with the same number of neuron blocks. As the complexity of problem increases, the performance of model3 and model5 improves.

5.3.2.4 Half adder truth problem

The XOR problem is a special case of half adder truth problem as discussed in chapter 4. In Figs. 5.21-5.24, the error convergence for the half adder truth problem is shown. Here, the convergence of model0, model1, model3, and model5 is better as compared to the model2, model4, and model6 for one and two neuron blocks. But as the number of neuron blocks are increased, the performance of model2, model4, and model6 improves as observed earlier for XOR and parity problems.

5.4 Concluding Remarks

In this chapter, the compensatory models were tested on various benchmark problems, and it is found that the performances of the compensatory models are better than STD. The generalization characteristics of compensatory models are much better than the STD. The compensatory models are computationally more efficient and converge faster too. Among the compensatory models, the model3 and the model5 seem to work better for a range of complex problems. The model0 and model1 are comparable and show better performance as compared to model2, model4, and model6.

For the spiral problem, it has been observed that the STD converges, but the compensatory models do not converge. This is expected for the problems where there is overlapping of different classes and thereby the proximity of different classes in the phase space. The compensatory models, being higher order models, create a higher order nonlinear boundary as compared to the STD, which creates a simple boundary. Therefore, the reparability in such cases using the compensatory models becomes difficult.

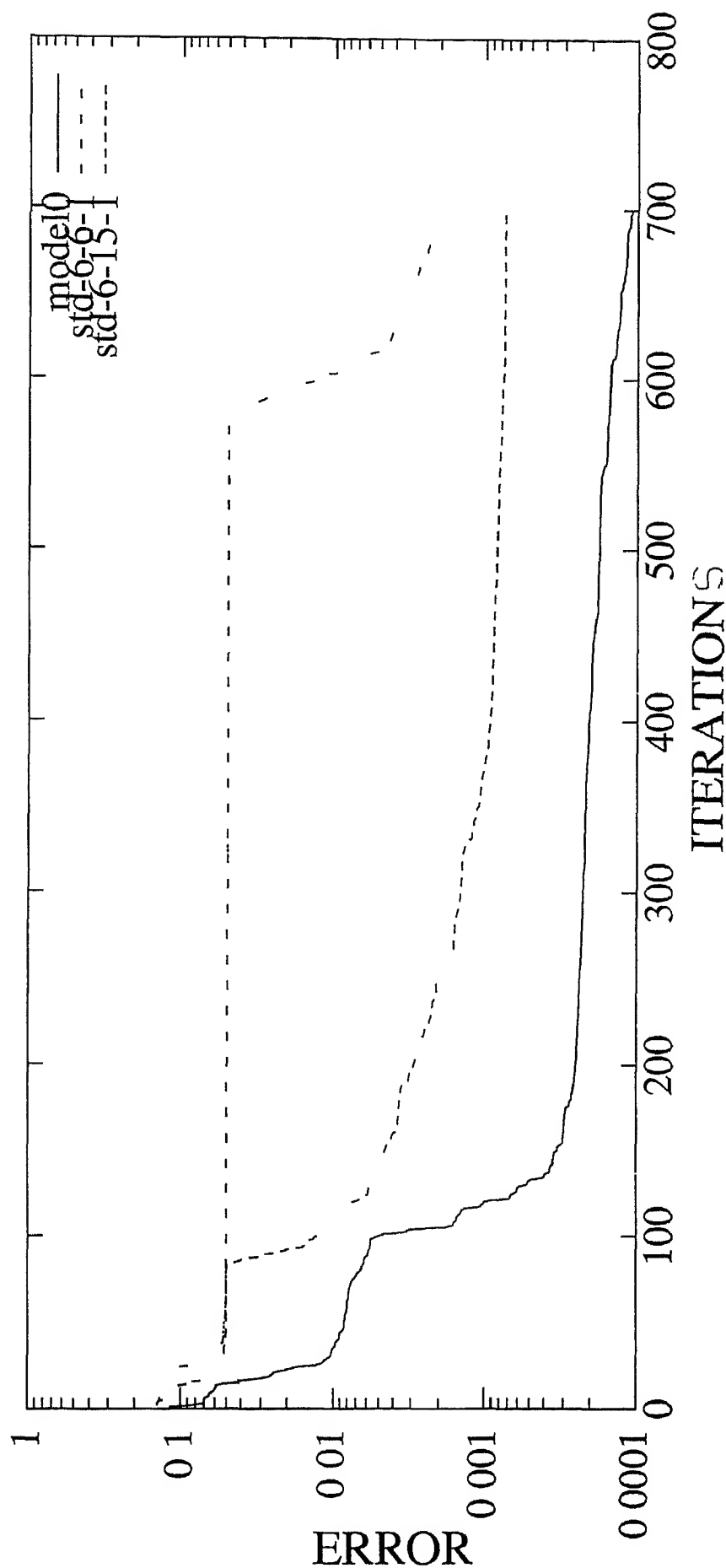


FIG 5 1a A comparison of error convergence for the STD architectures and the compensatory model0 for $\sin(x) \cdot \sin(y)$ problem

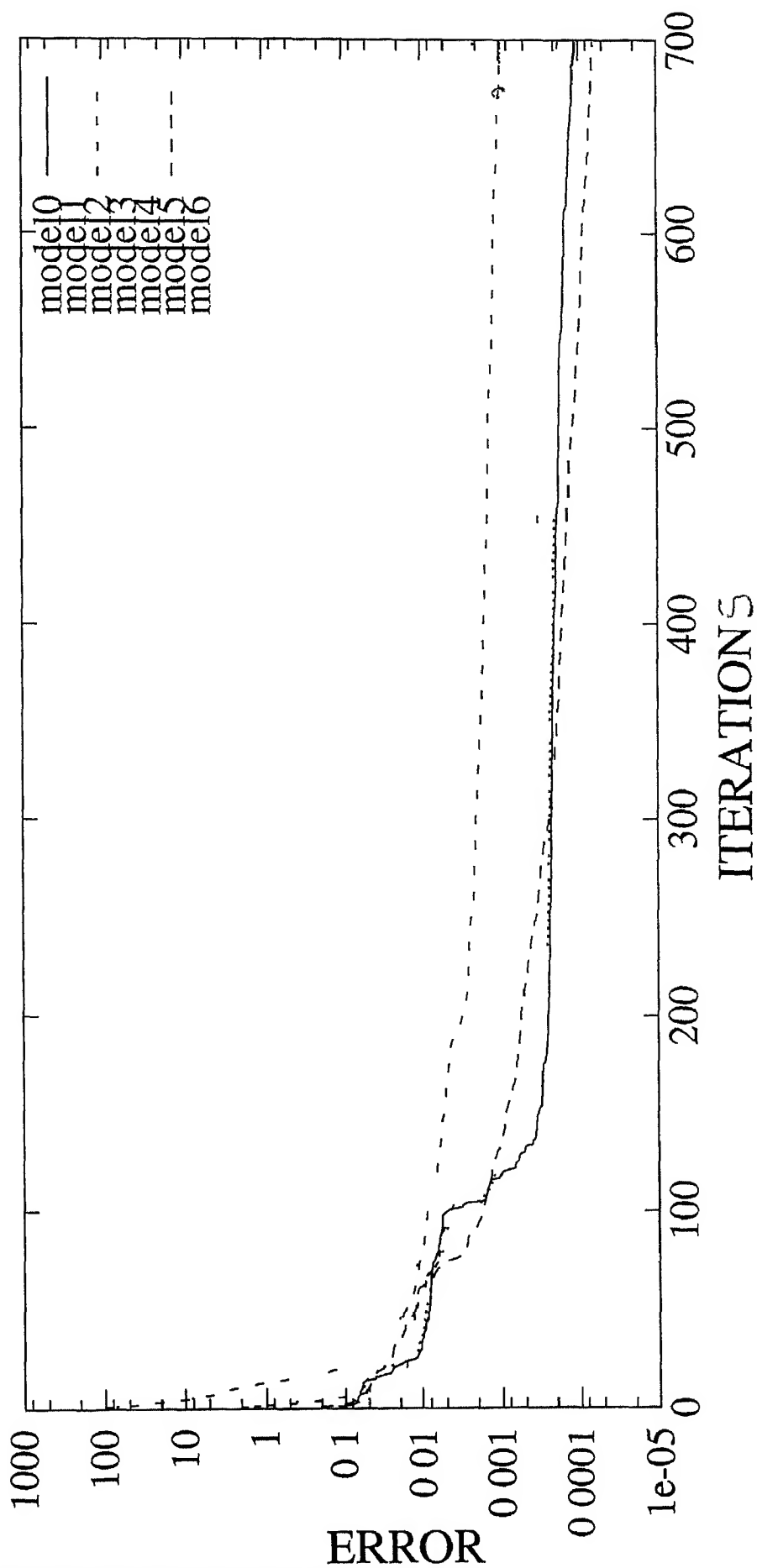


FIG 5 1b Convergence error for $\sin(x) \cdot \sin(y)$ problem on training set for compensatory models using SSCGA learning

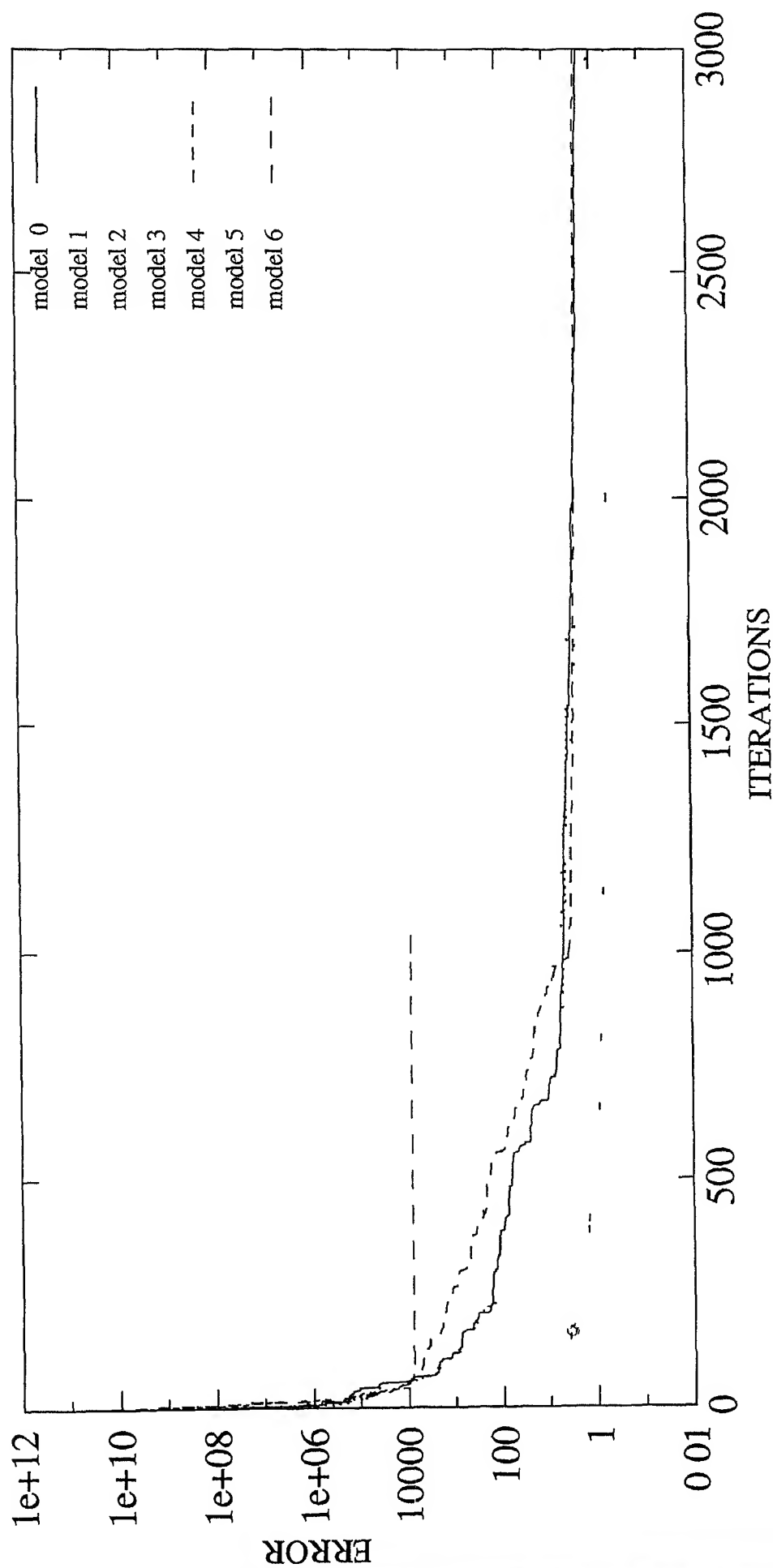


FIG 5 2 Error convergence for the functional during training

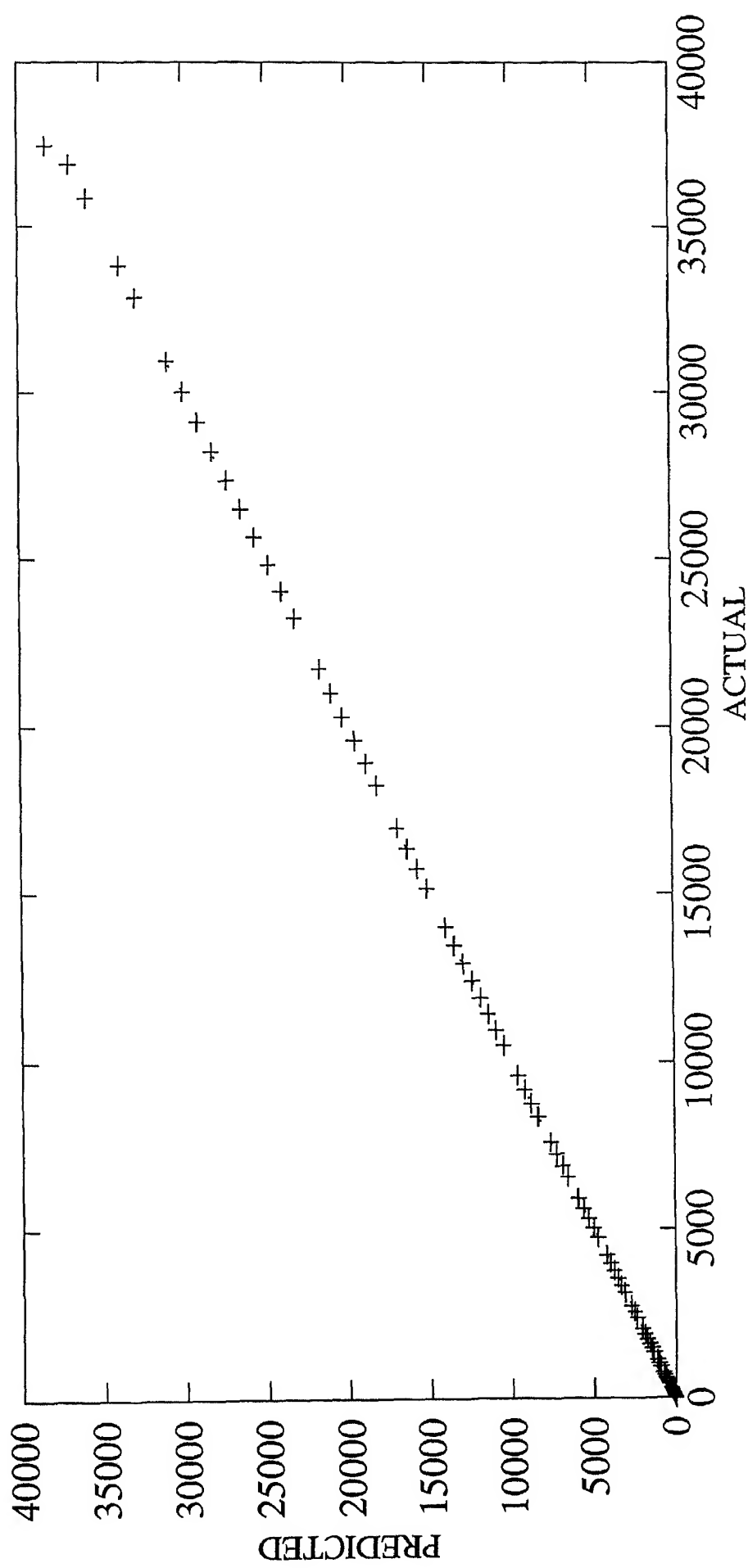


FIG 5 3a Comparison of Actual Vs Predicted values obtained using model0 with SSCGA learning

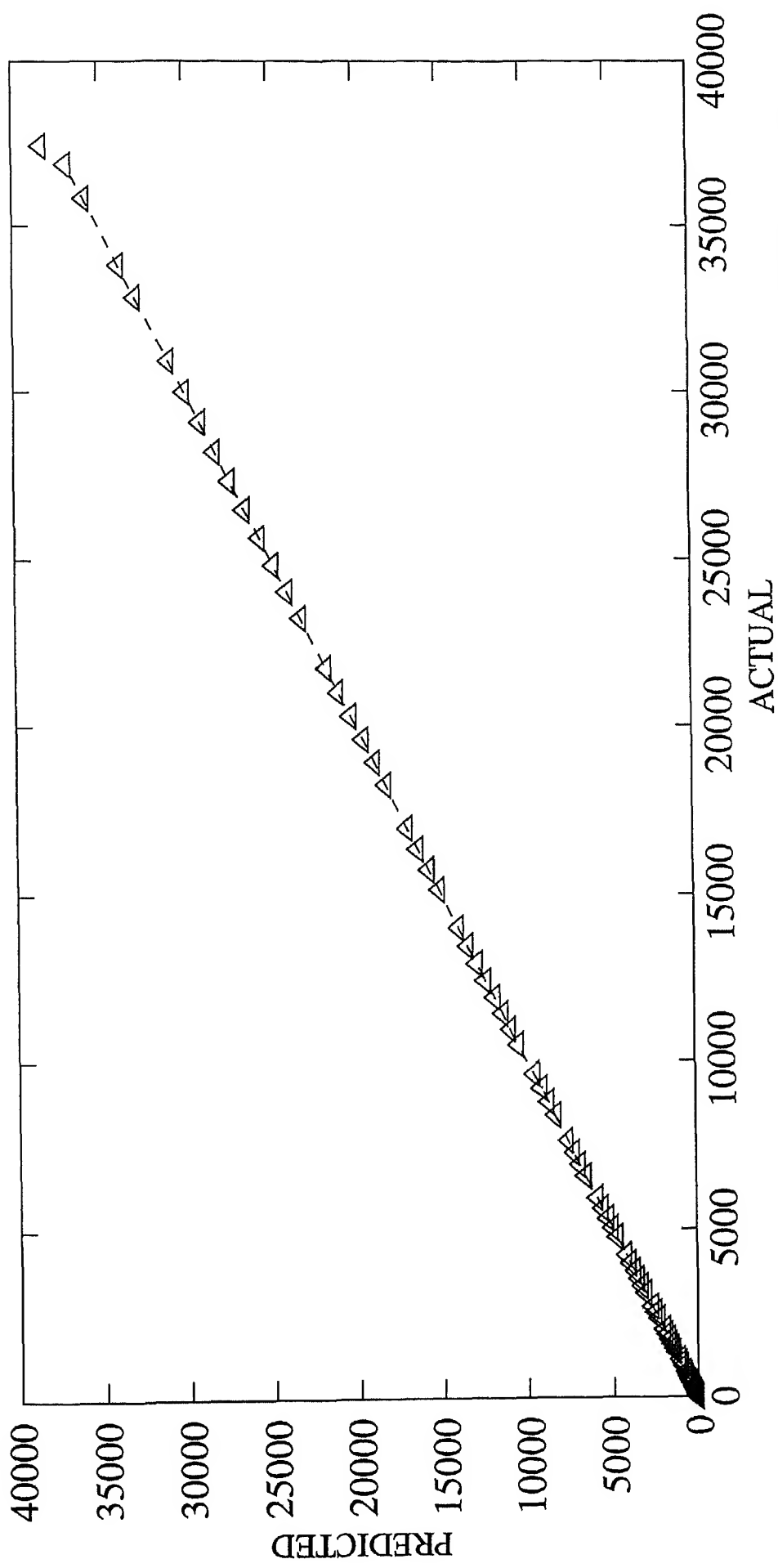


FIG 5 3b Comparison of Actual V_s Predicted values obtained using Model1 with SSCGA learning

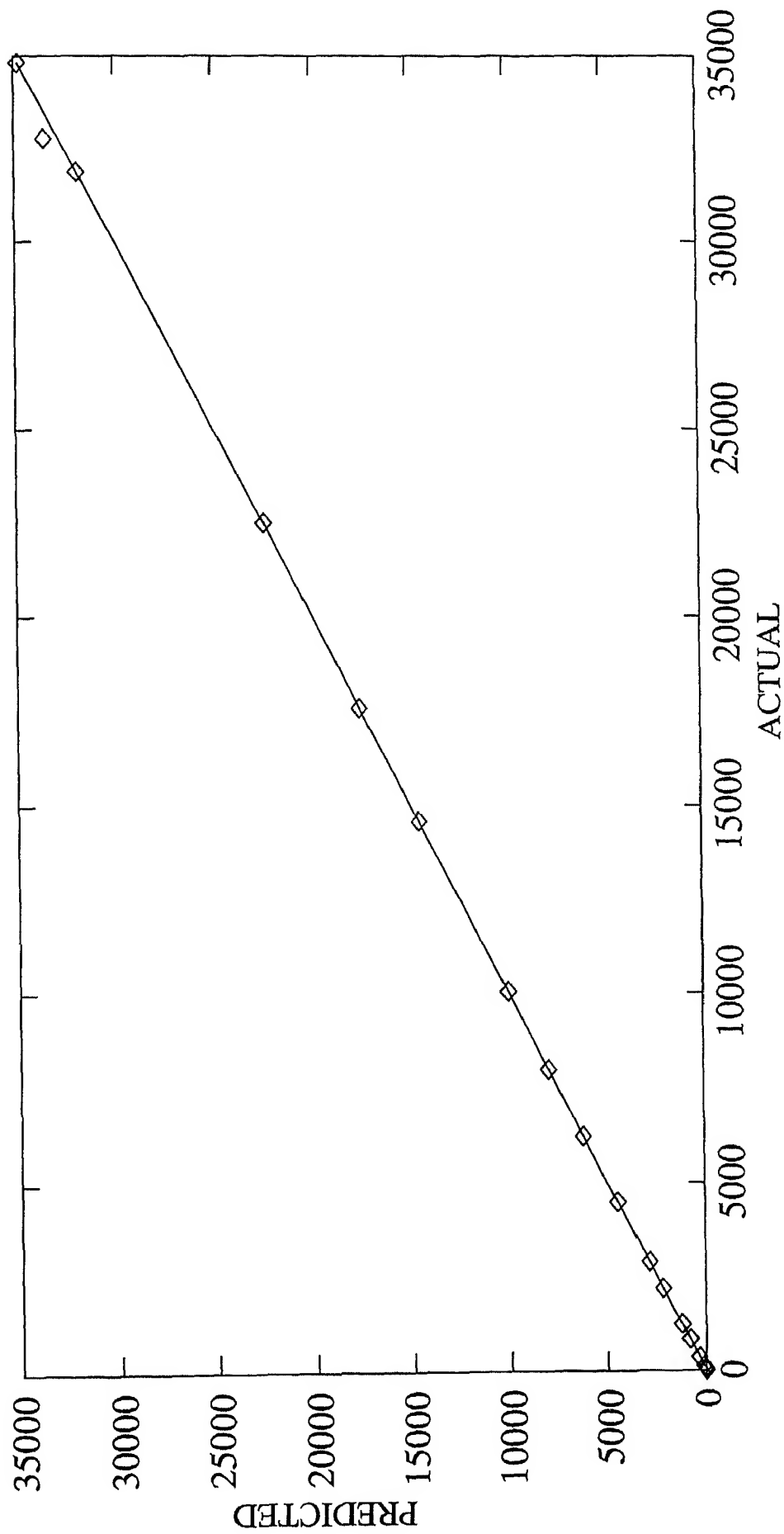


FIG 5 4a Comparison of Actual Vs Predicted values obtained using Model0 with SSCGA learning

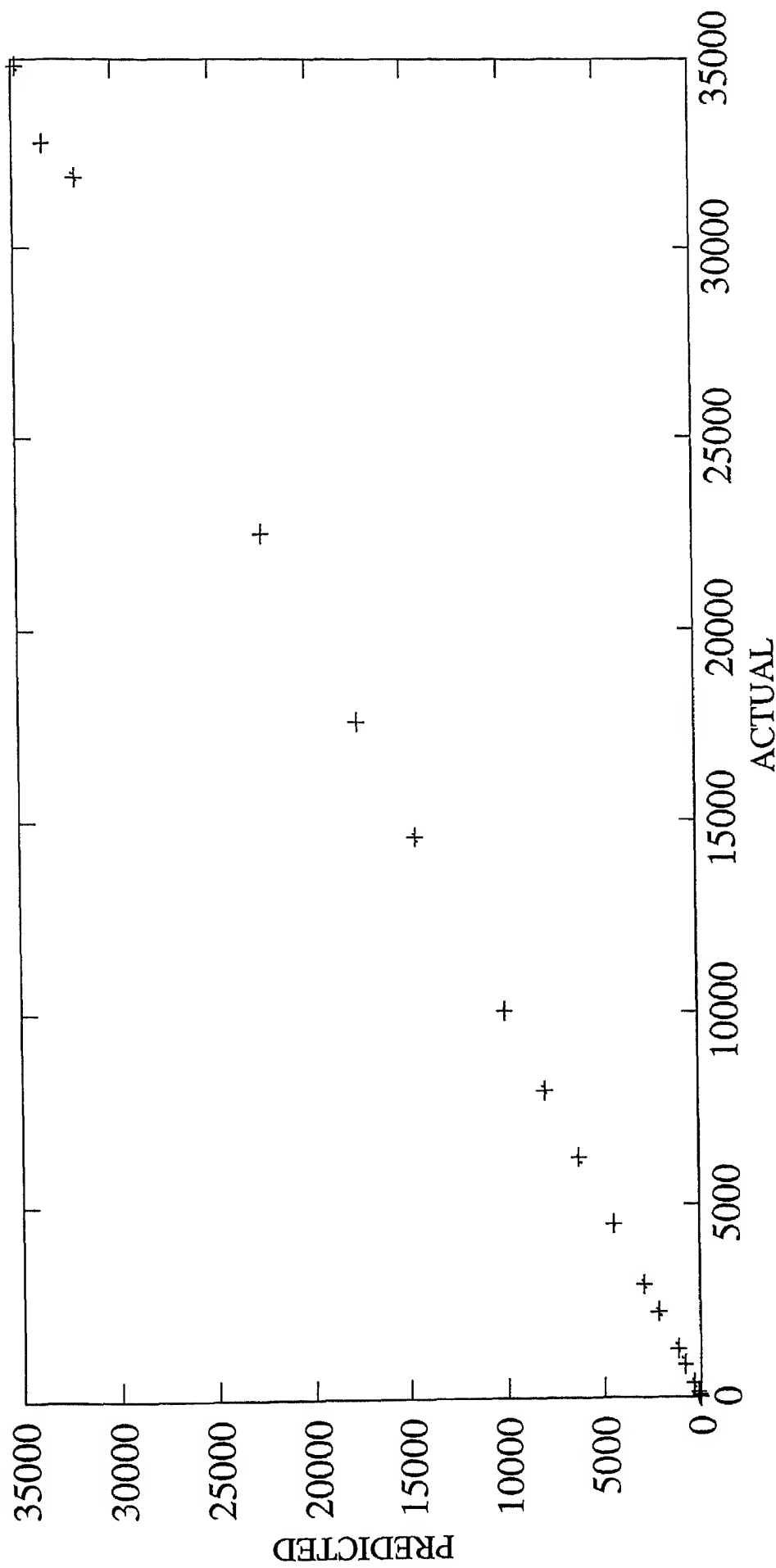


FIG 5 4b Comparison of Actual Vs Predicted values obtained using Model1 with SSCGA learning

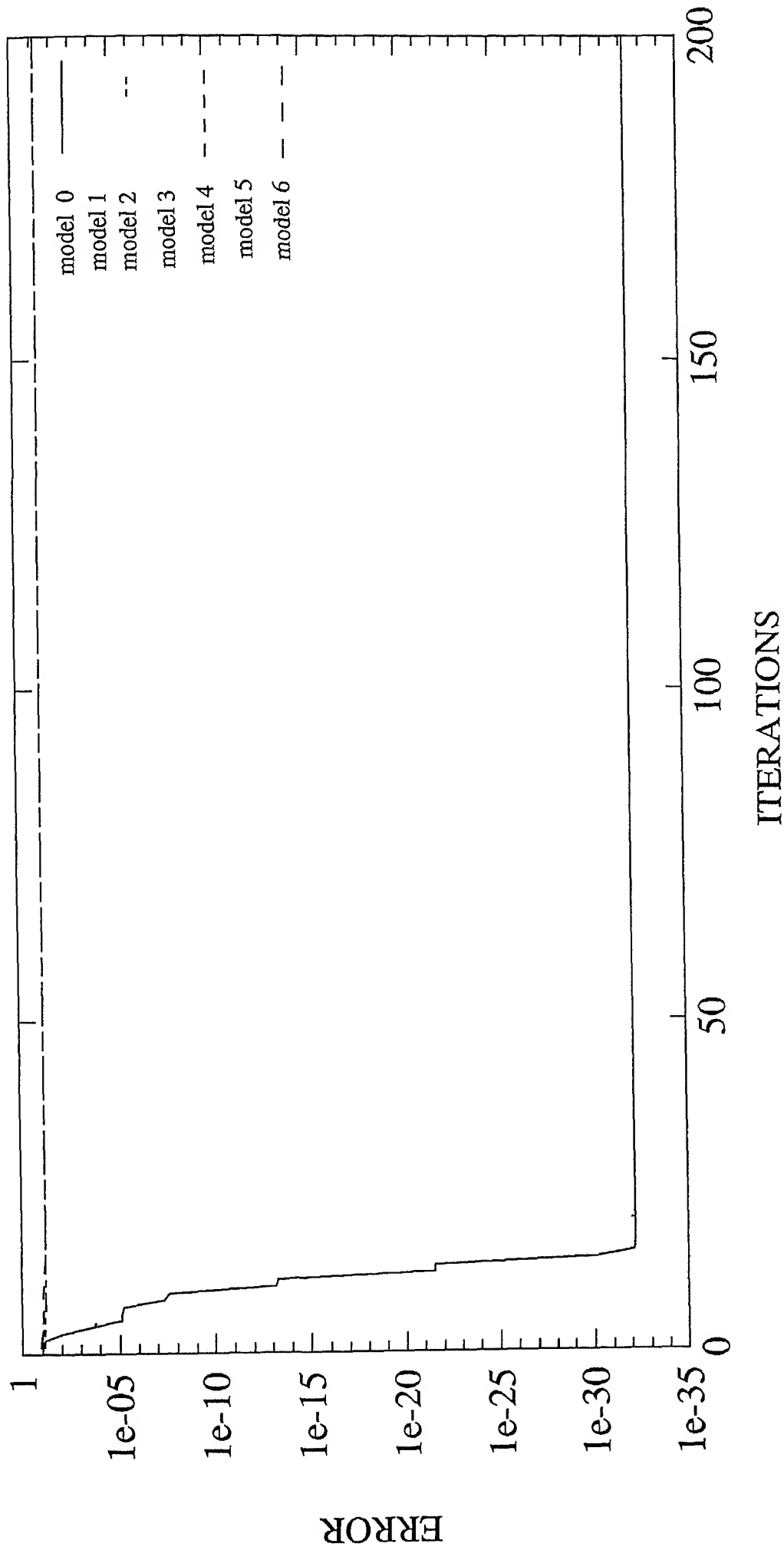


FIG 5 5 Error plot of XOR for CNNA with one Neuron

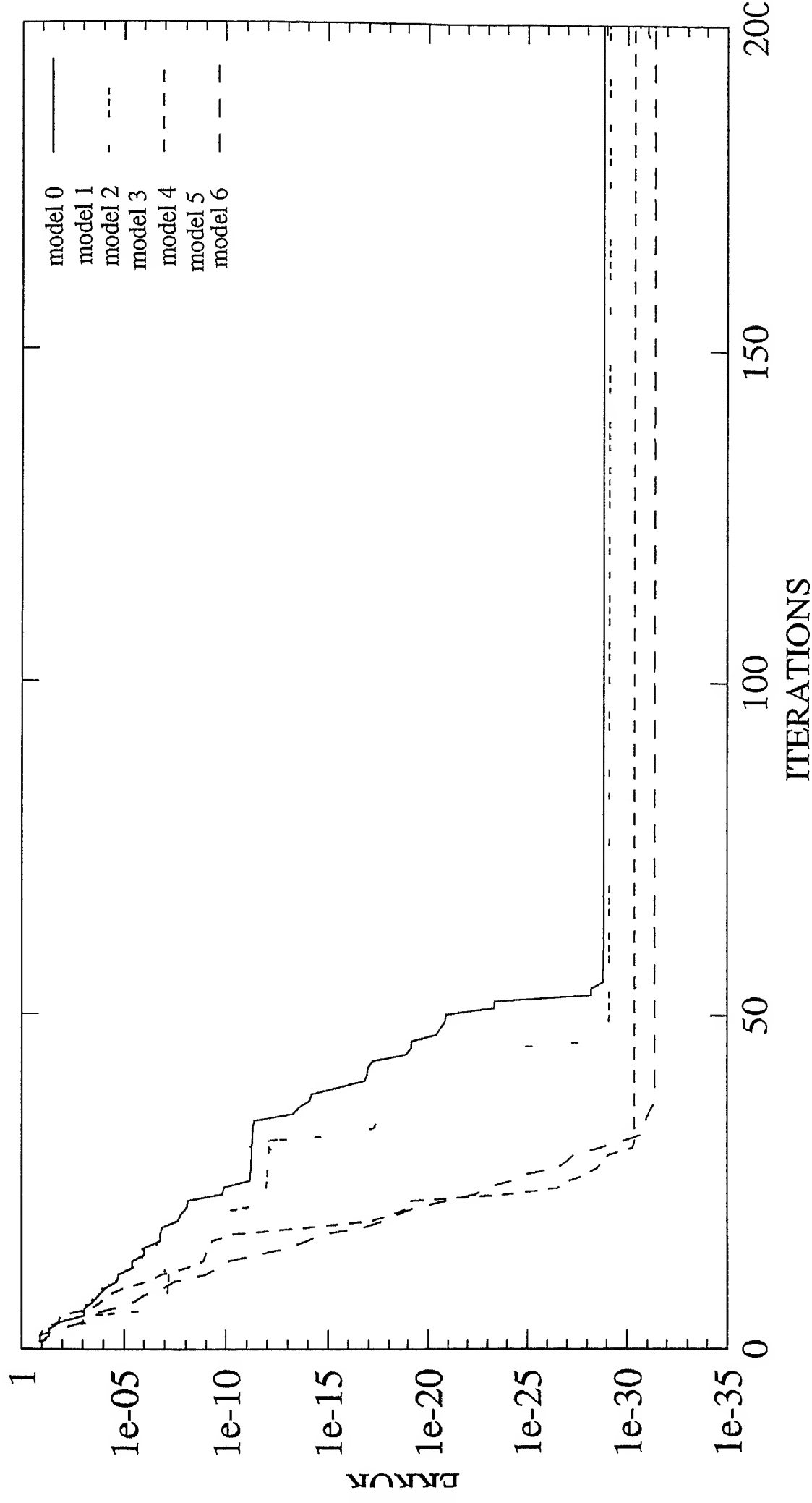


FIG 5 6 Error plot of XOR problem for CNNA with 2 Neurons

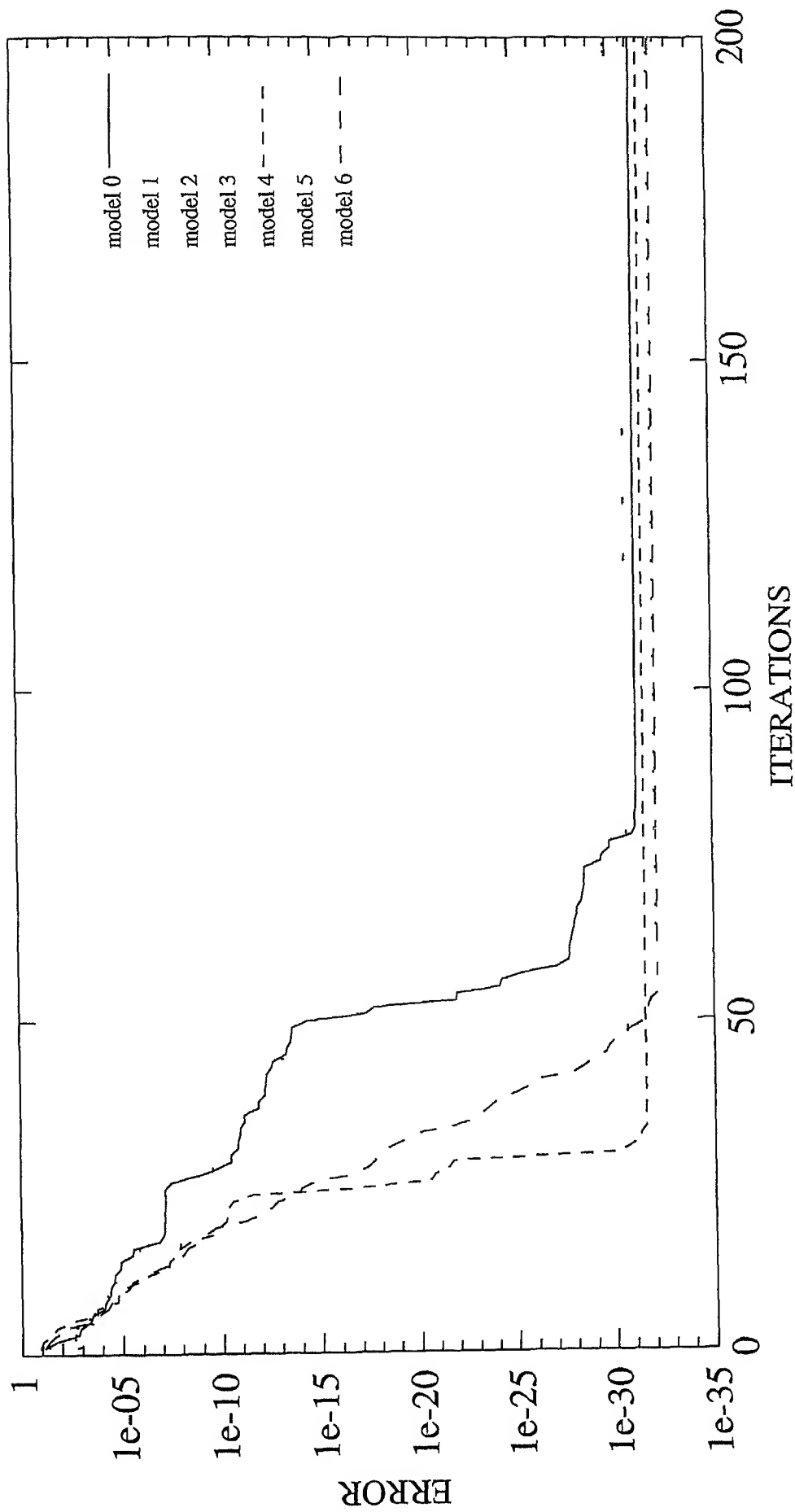


FIG 5 7 Error plot of XOR Problem for CNNA with 3 Neurons

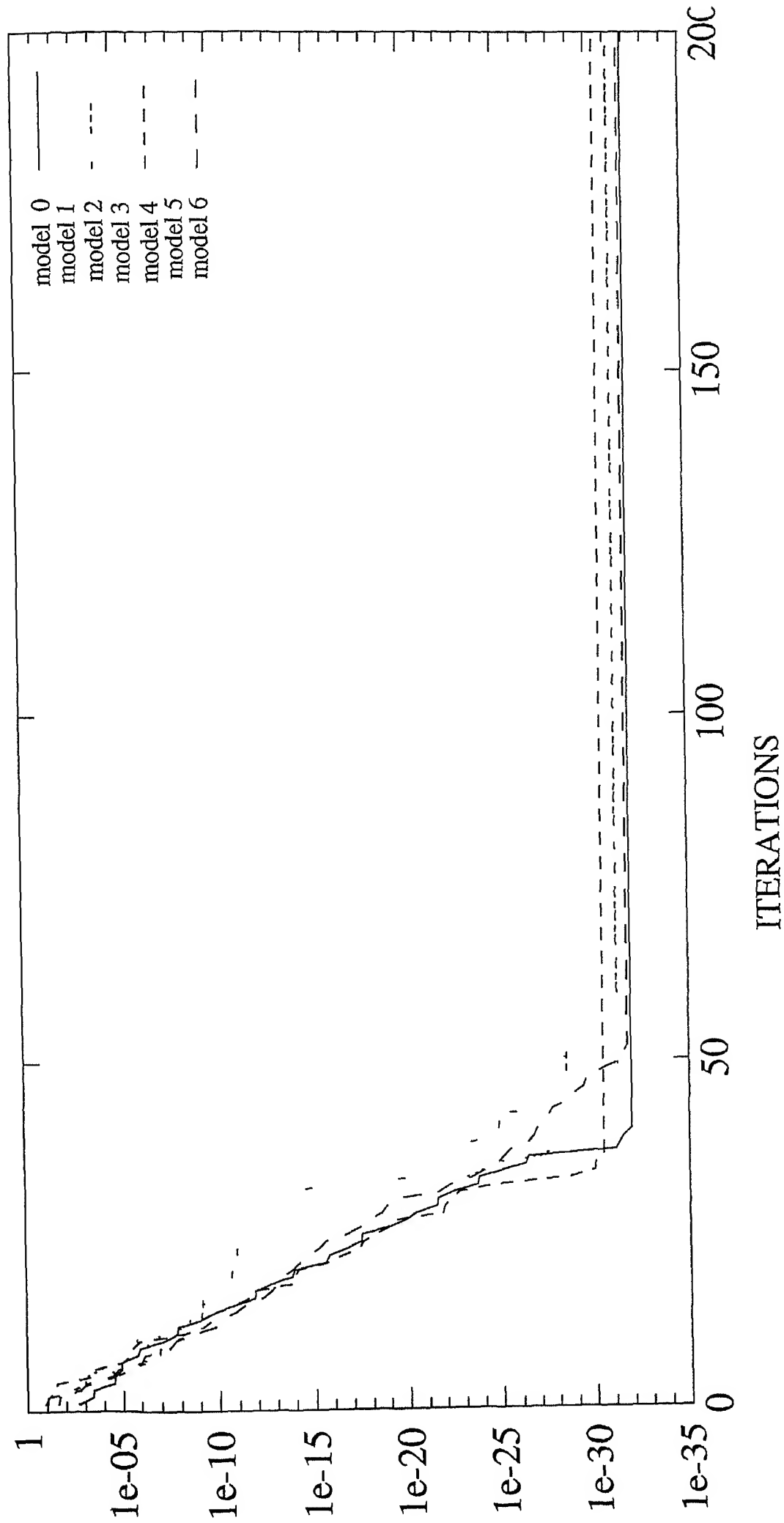


FIG 5 8 Error Plot of XOR problem for CNNA with 4 Neurons

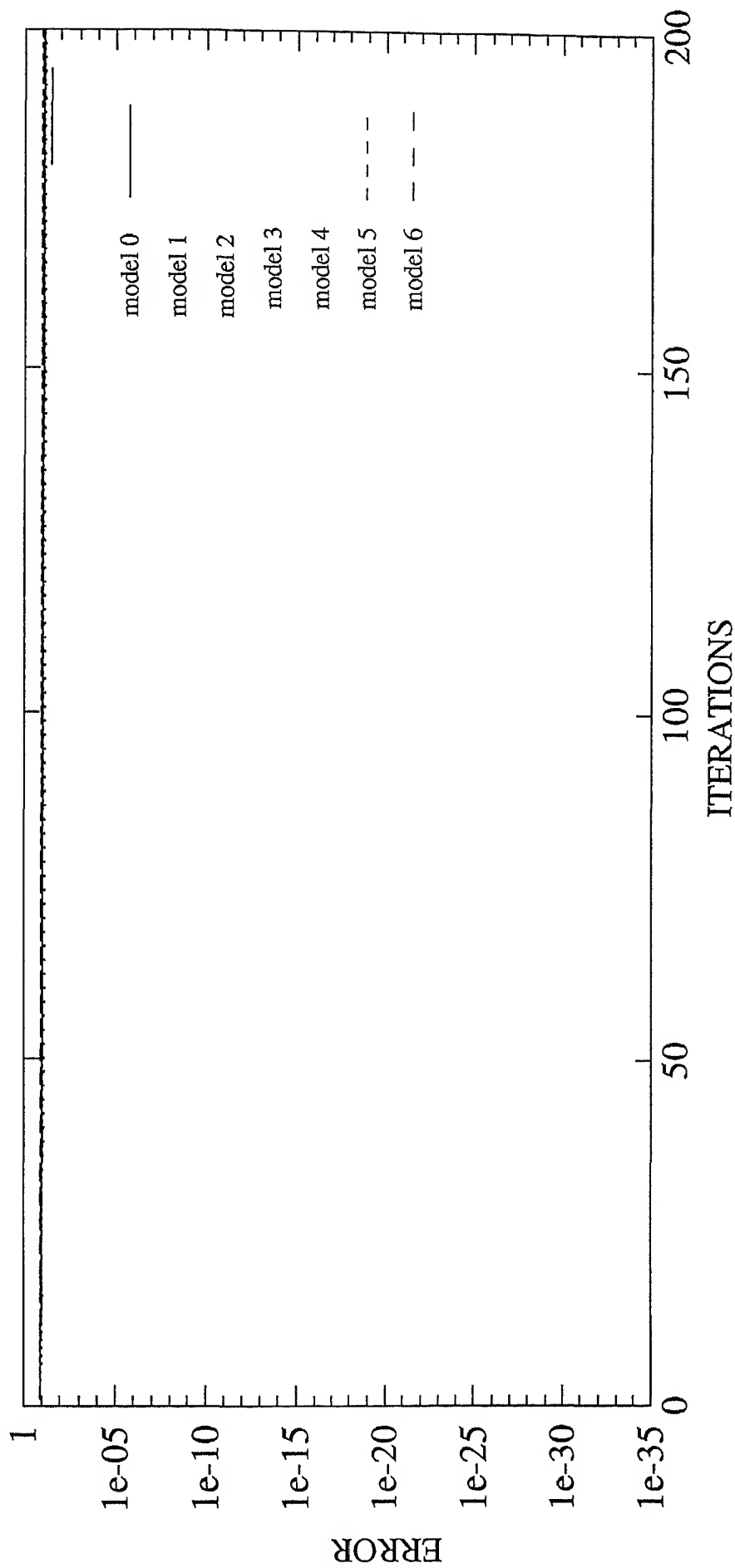


FIG 5 9 Error plot of 4-bit parity problem for CNNA with one Neuron

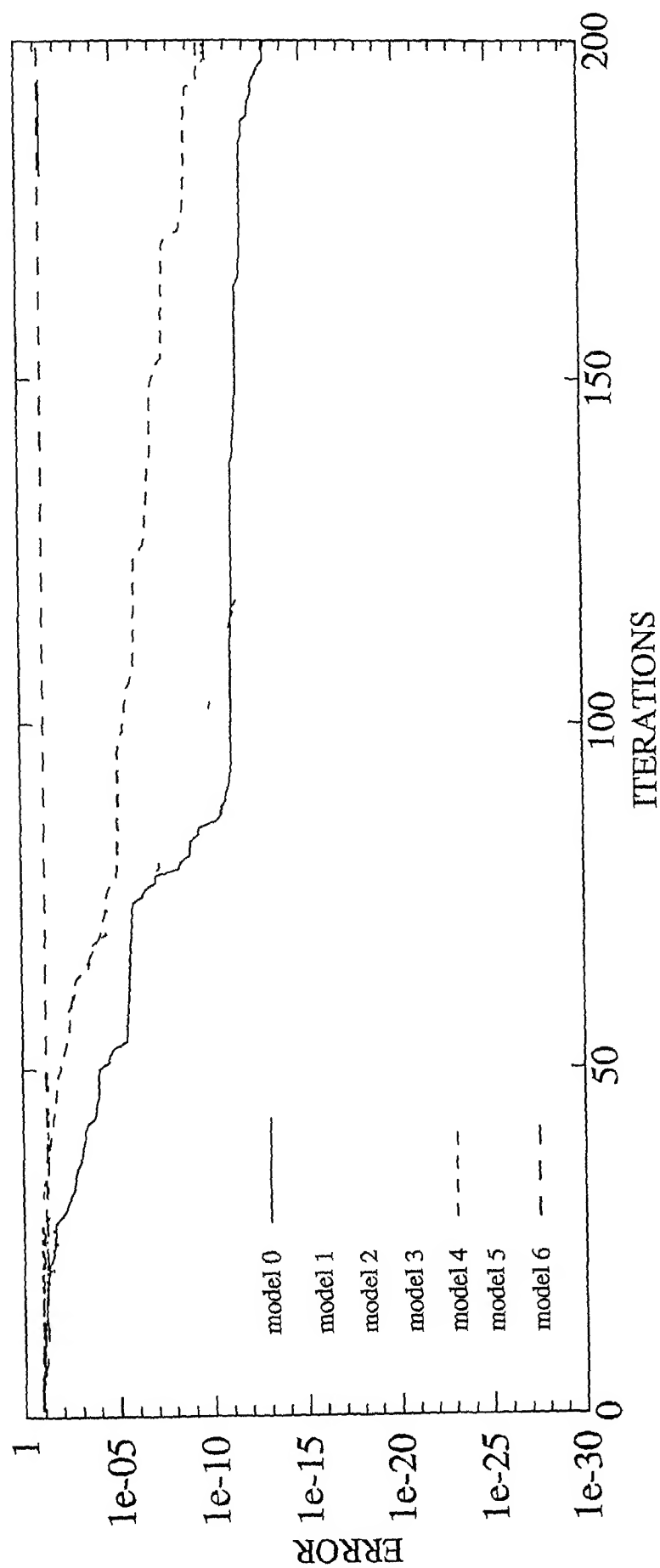


FIG 5 10 Error plot of 4 - bit Parity problem for CNN with 2 Neurons

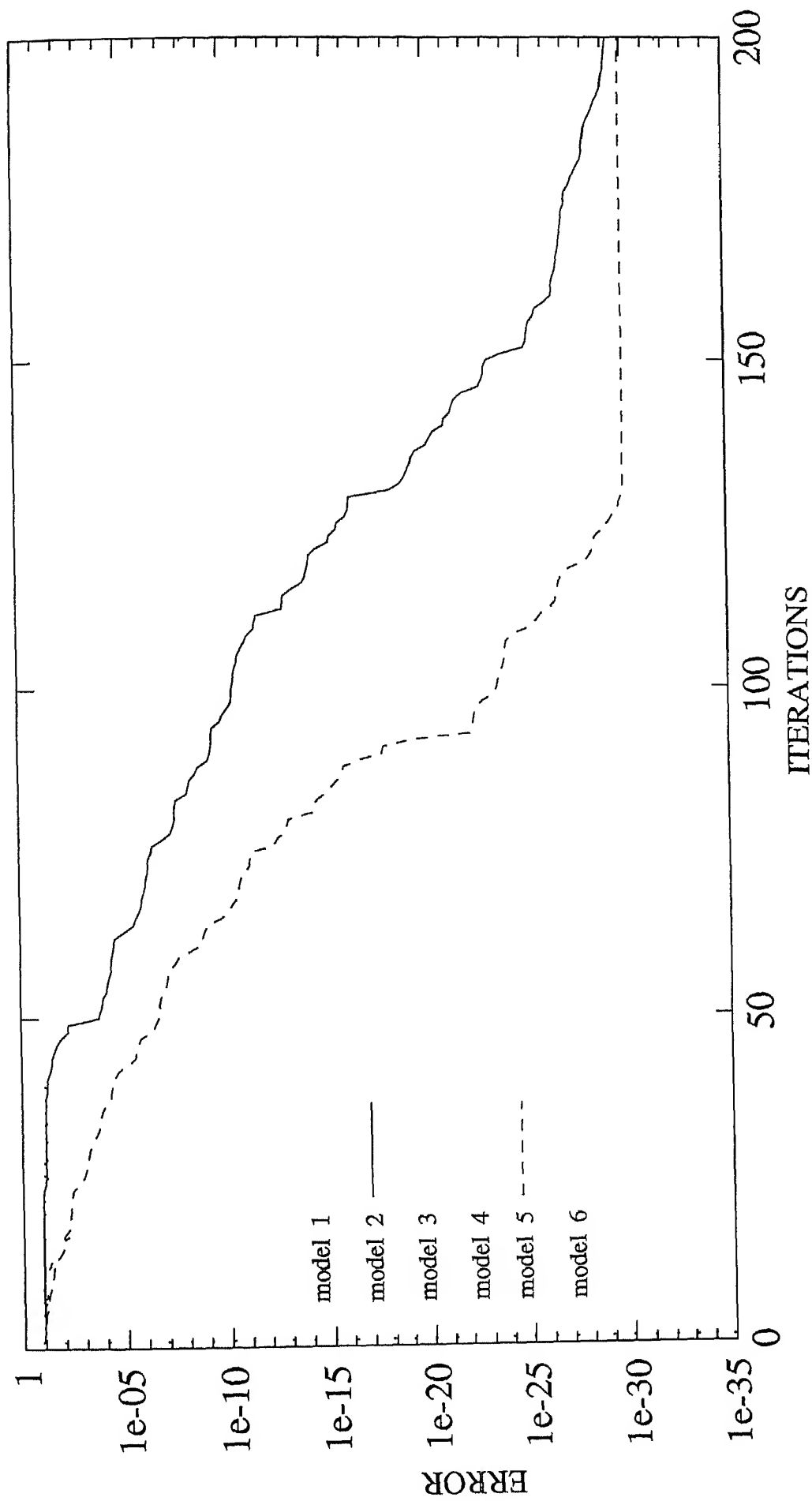


FIG 5 11 Error plot of 4 - bit Parity problem for CNNA with 3 Neurons

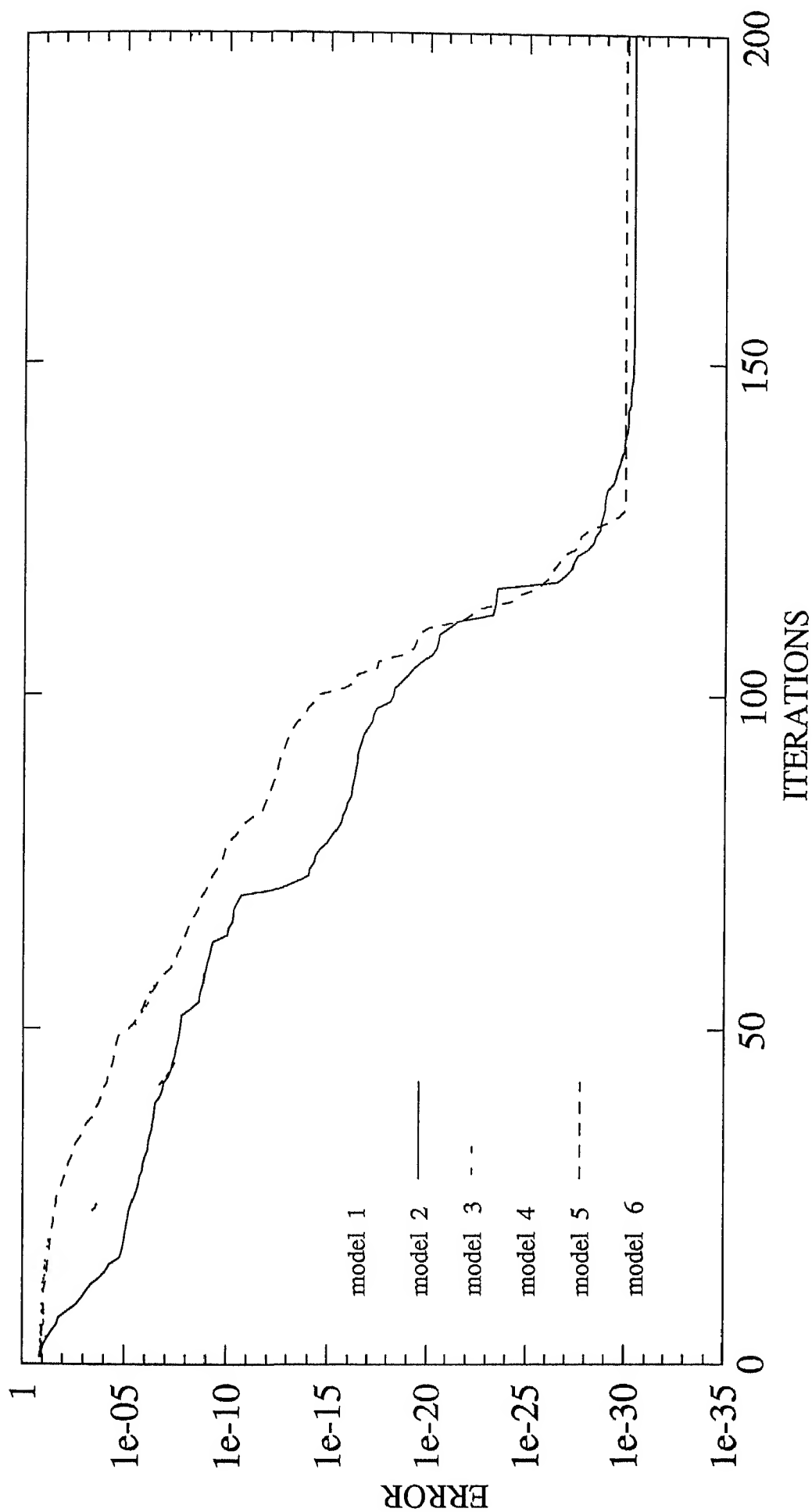


FIG 5 12 Error plot of 4 - bit Parity problem for CNNA with 4 neurons

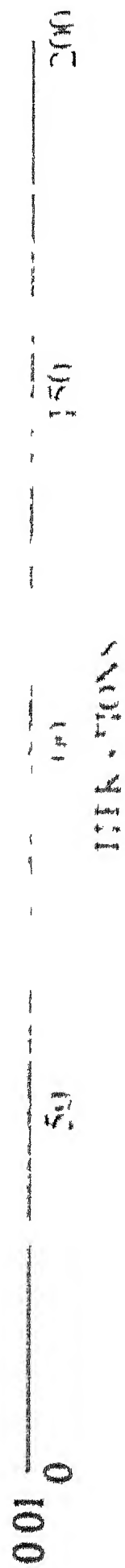
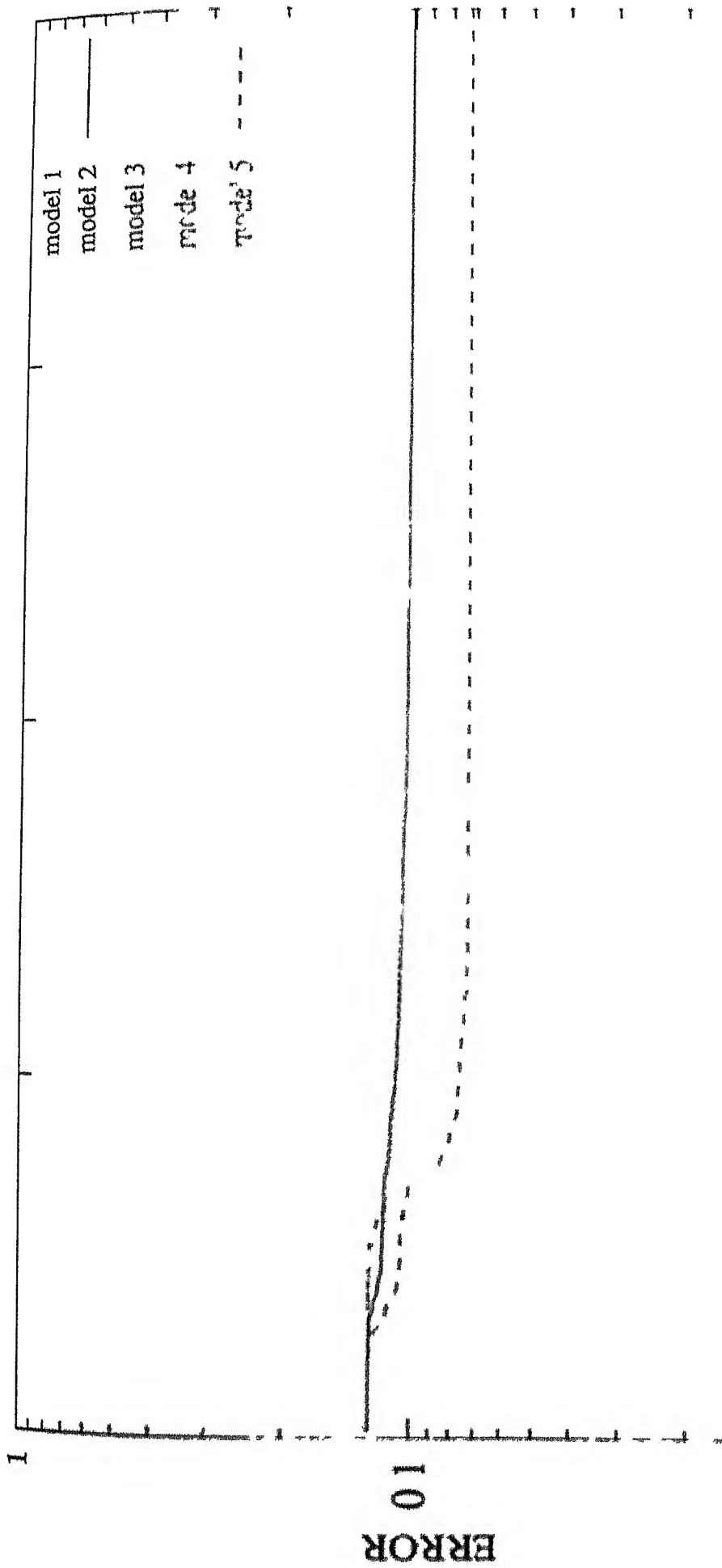


FIG. 2. Error vs. Iterations for five models.

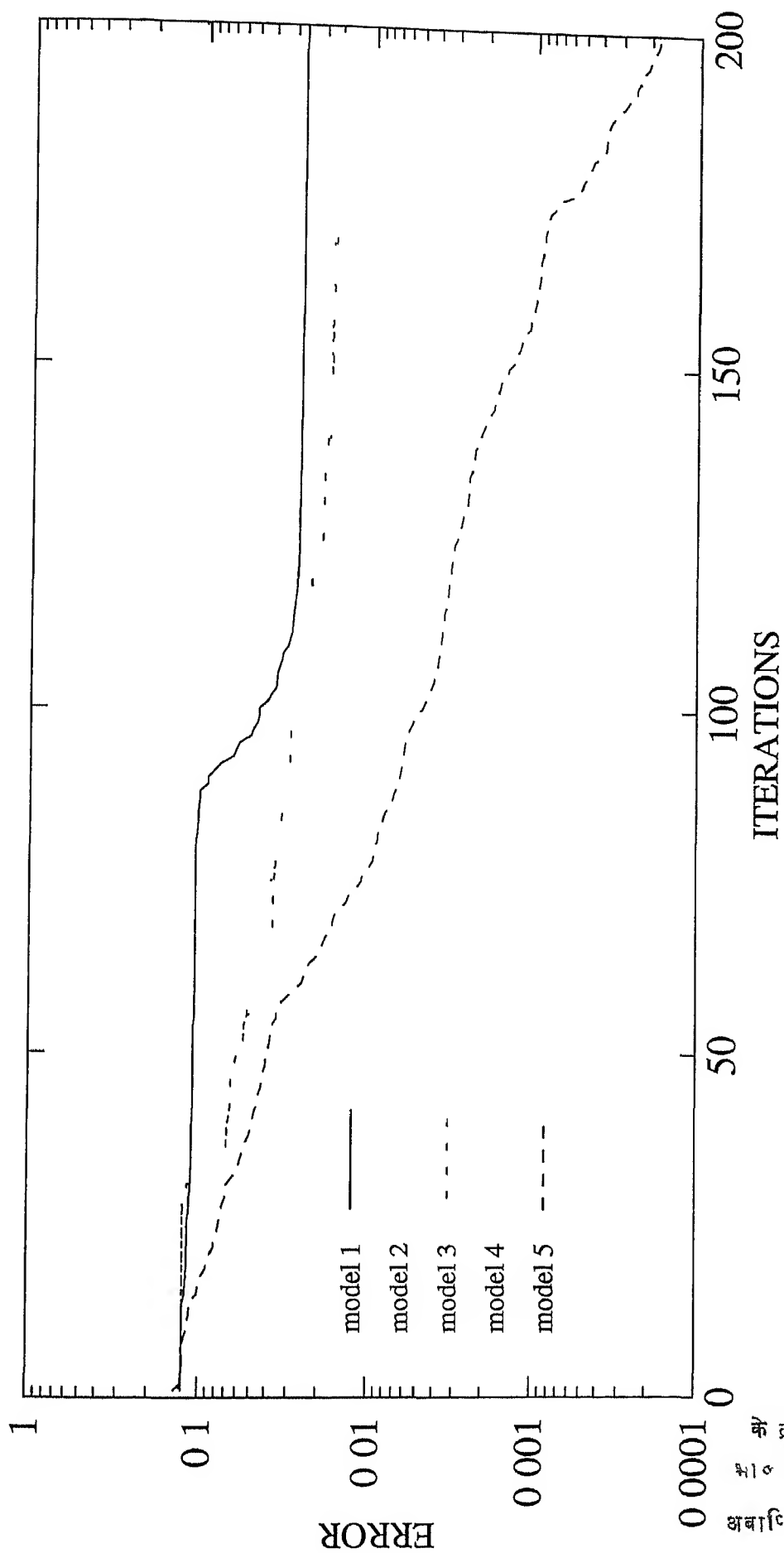


FIG 5 14 Error plot of 5 - bit Parity problem for CNNA with 2 neurons

के द्रीय पुस्तकालय
भा. 513 1 फानपुर
अबाप्ति-क्र. 9193316

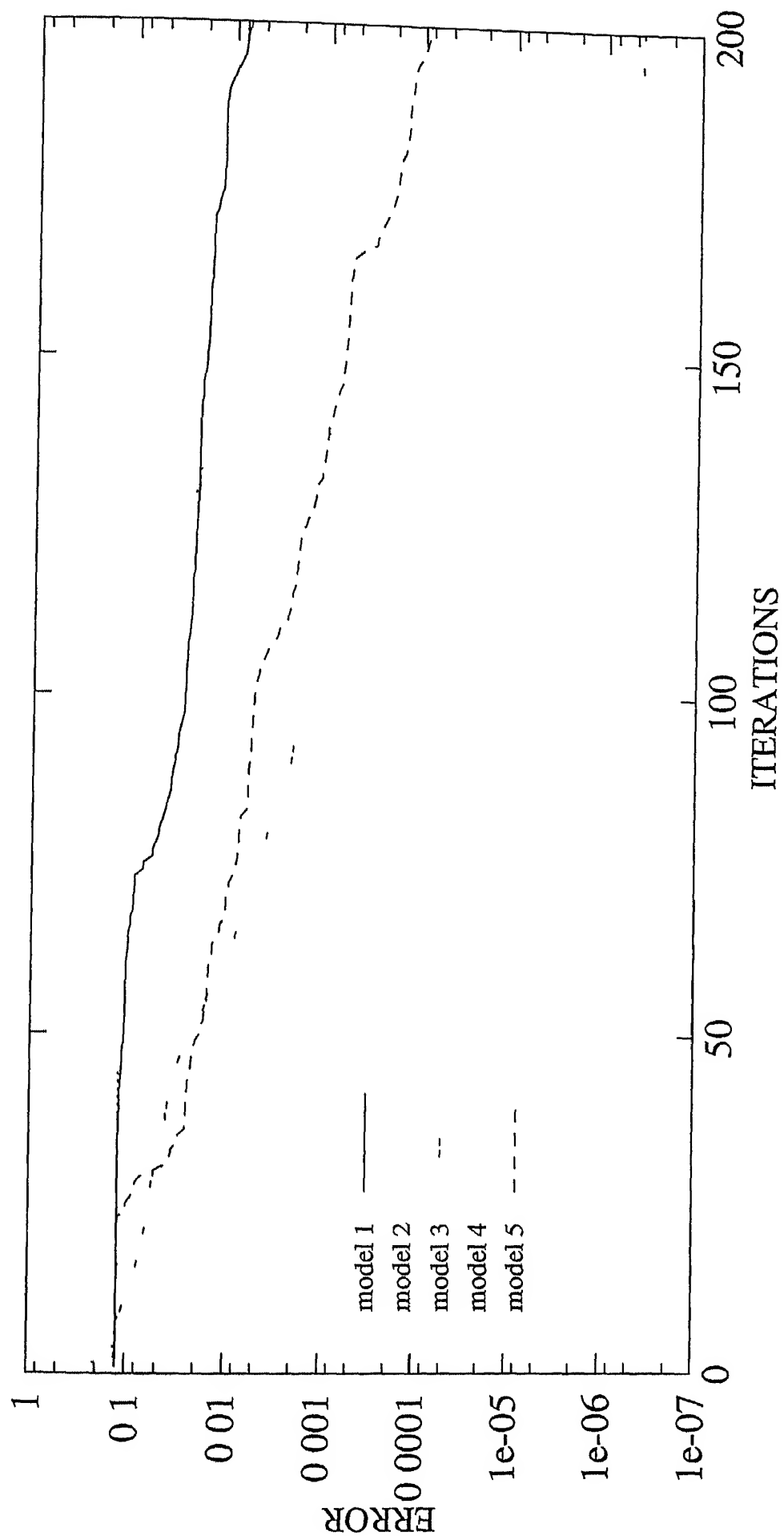


FIG 5 15 Error plot of 5-bit Parity problem for CNNA with 3 neurons

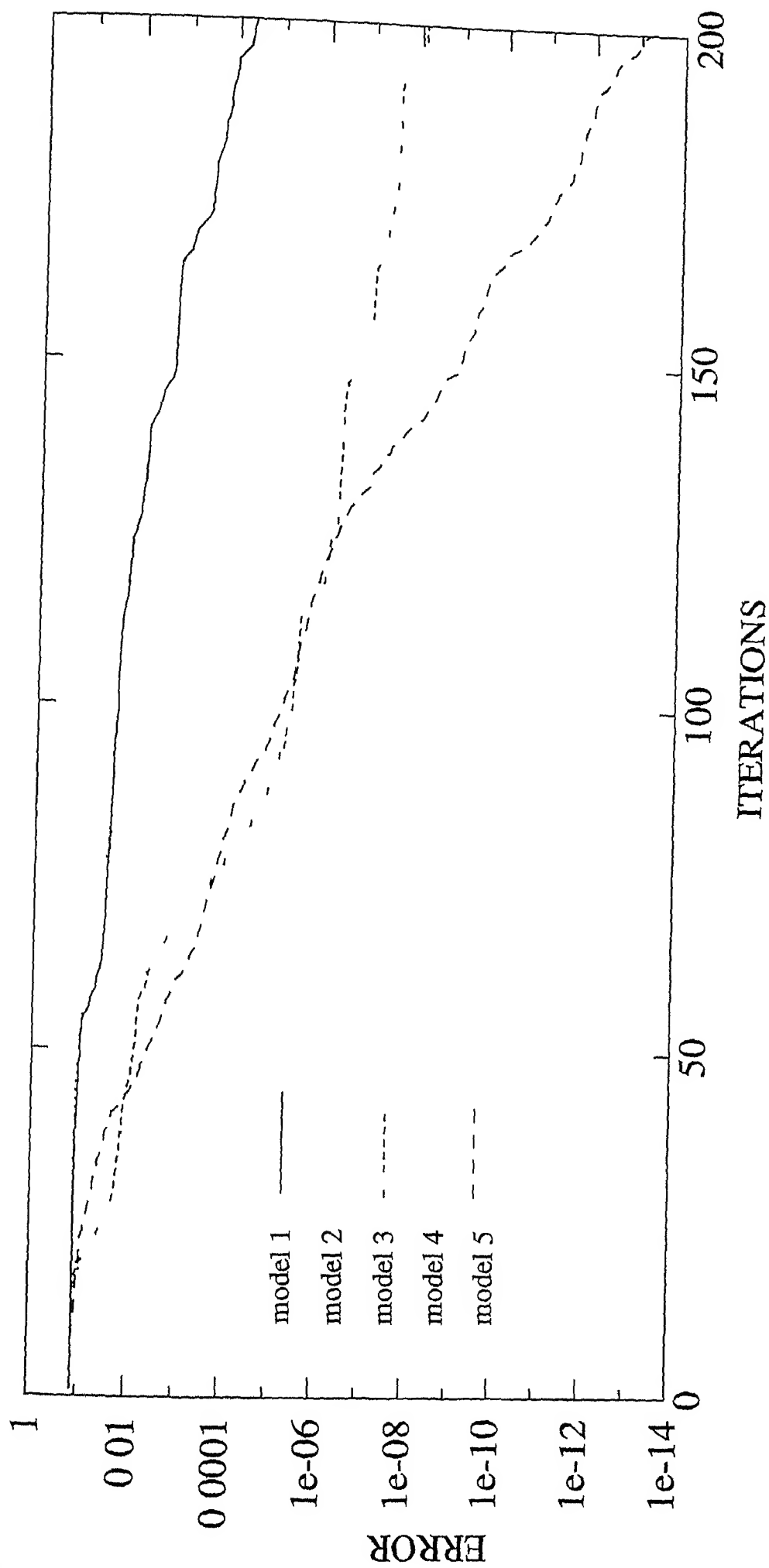


FIG 5 16 Error plot of 5 - bit Parity problem for CNNA with 4 neurons

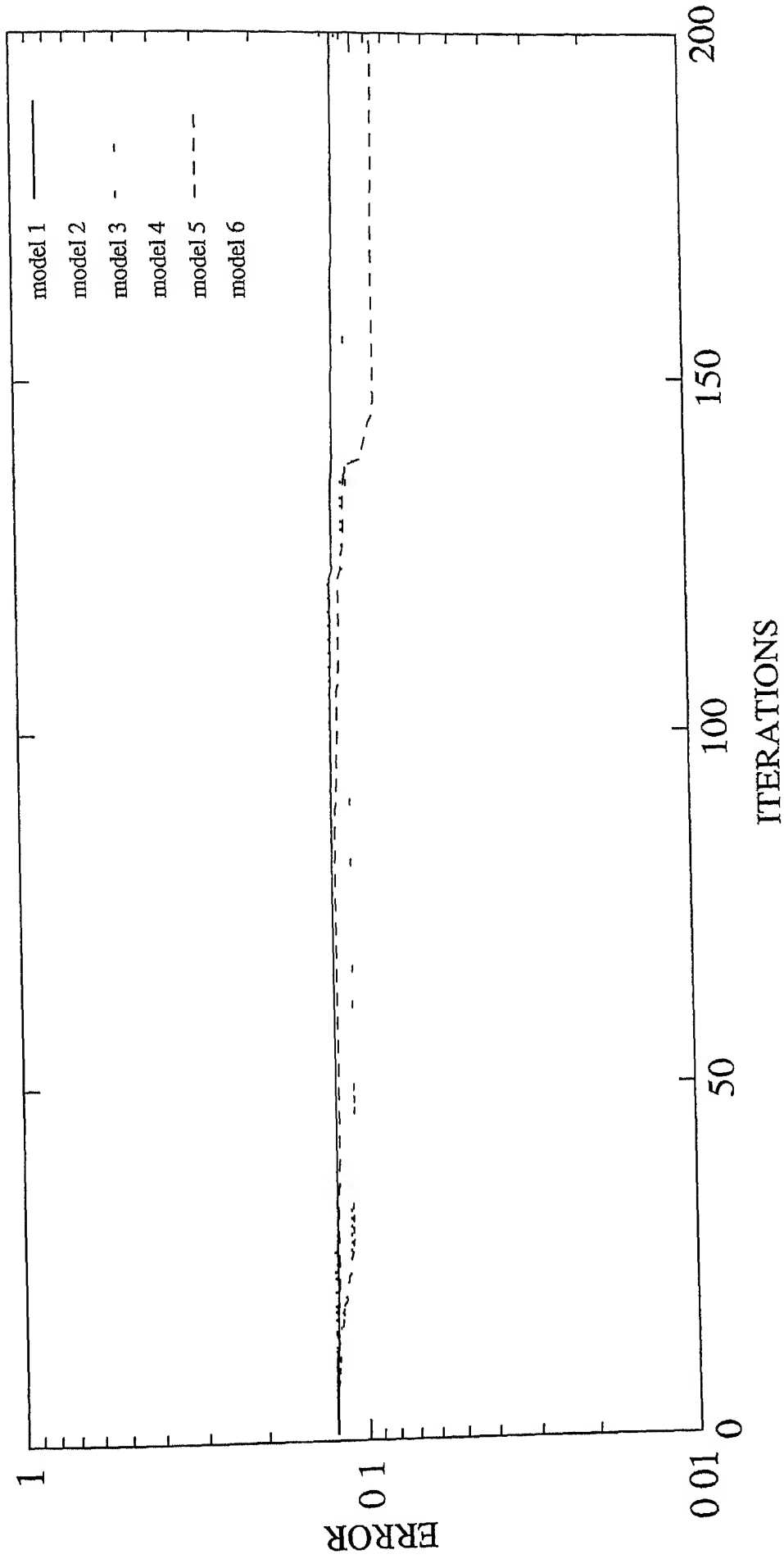


FIG 5 17 Error plot of 6 - bit Parity problem for CNNA with 1 neuron

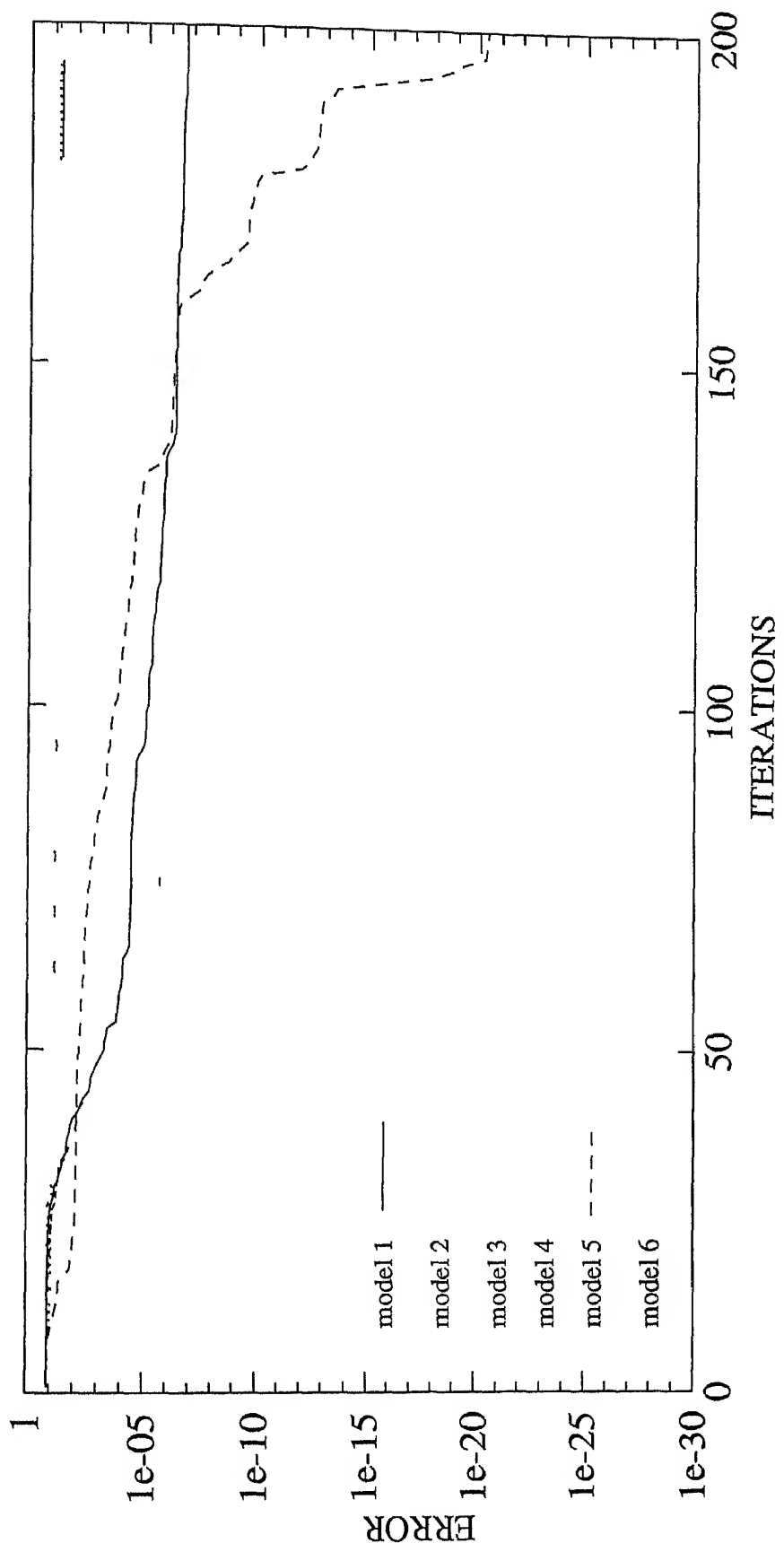


FIG 5 18 Error plot of 6 - bit Parity problem for CNNA with 2 neurons

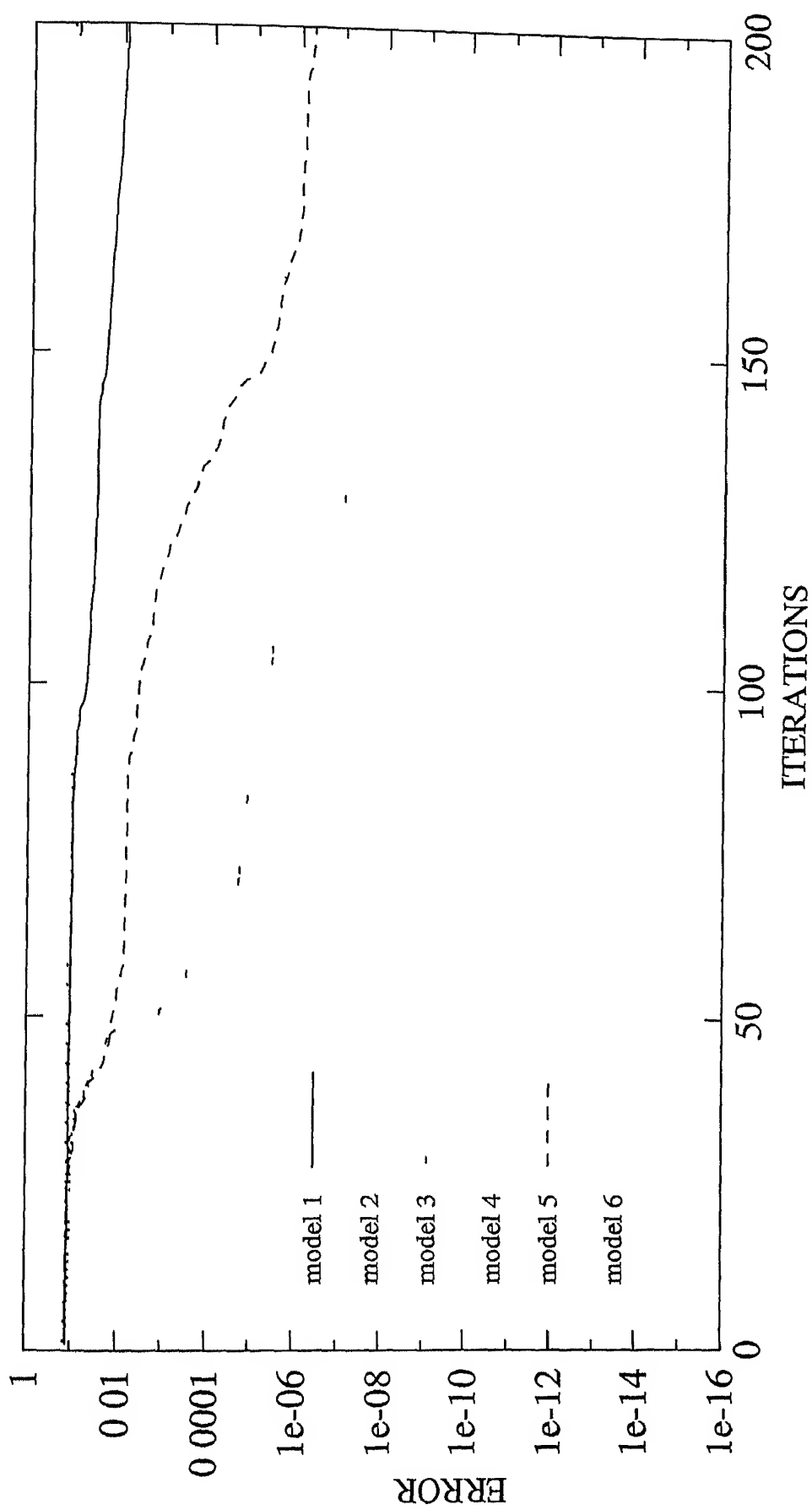


FIG 5 19 Error plot of 6 - bit Parity problem for CNNA with 3 neurons

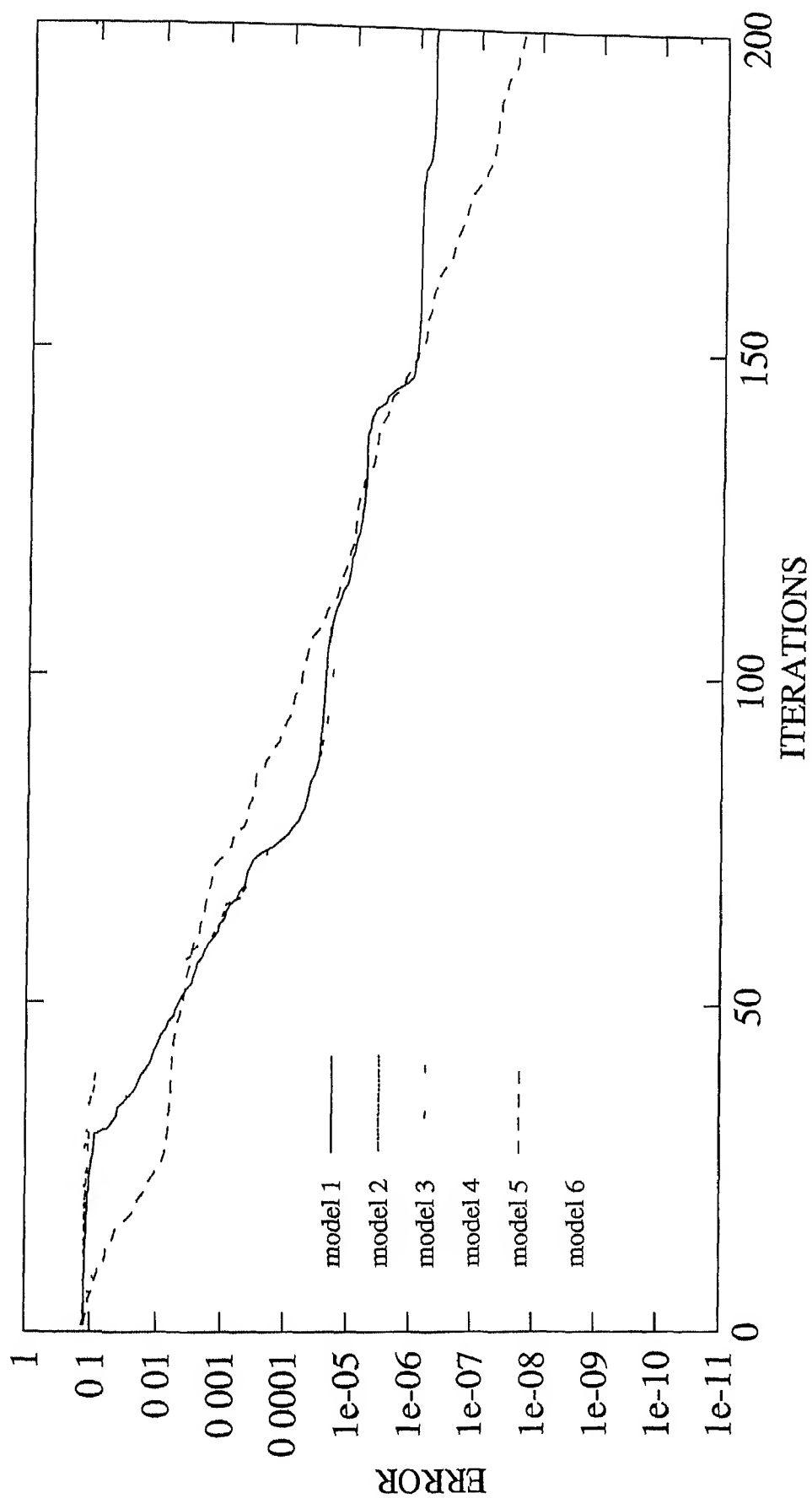


FIG 5 20 Error plot of 6 - bit parity problem for CNNA with 4 neurons

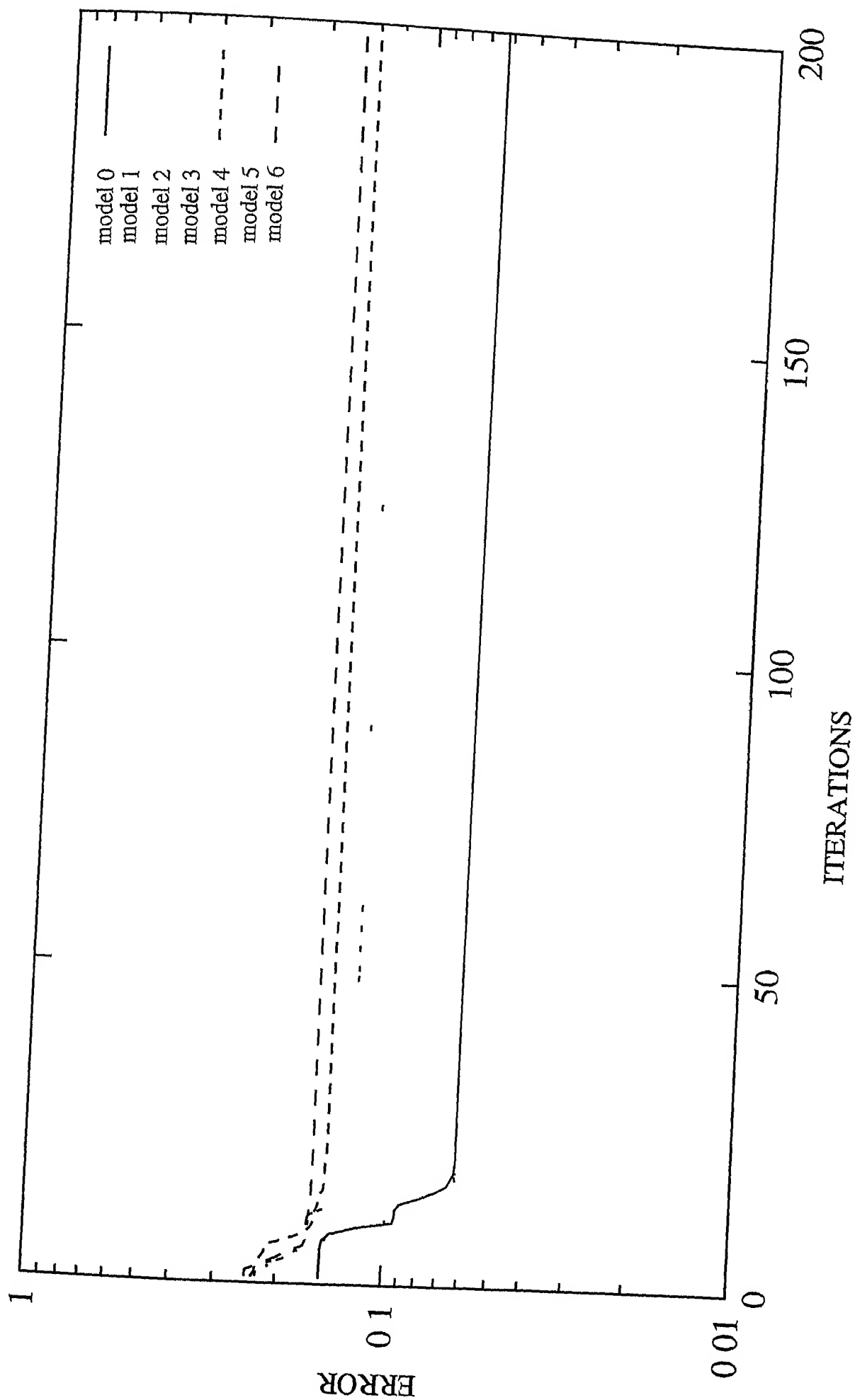


FIG 5 21 Error plot of Half adder truth problem for CNNA with 1 neuron

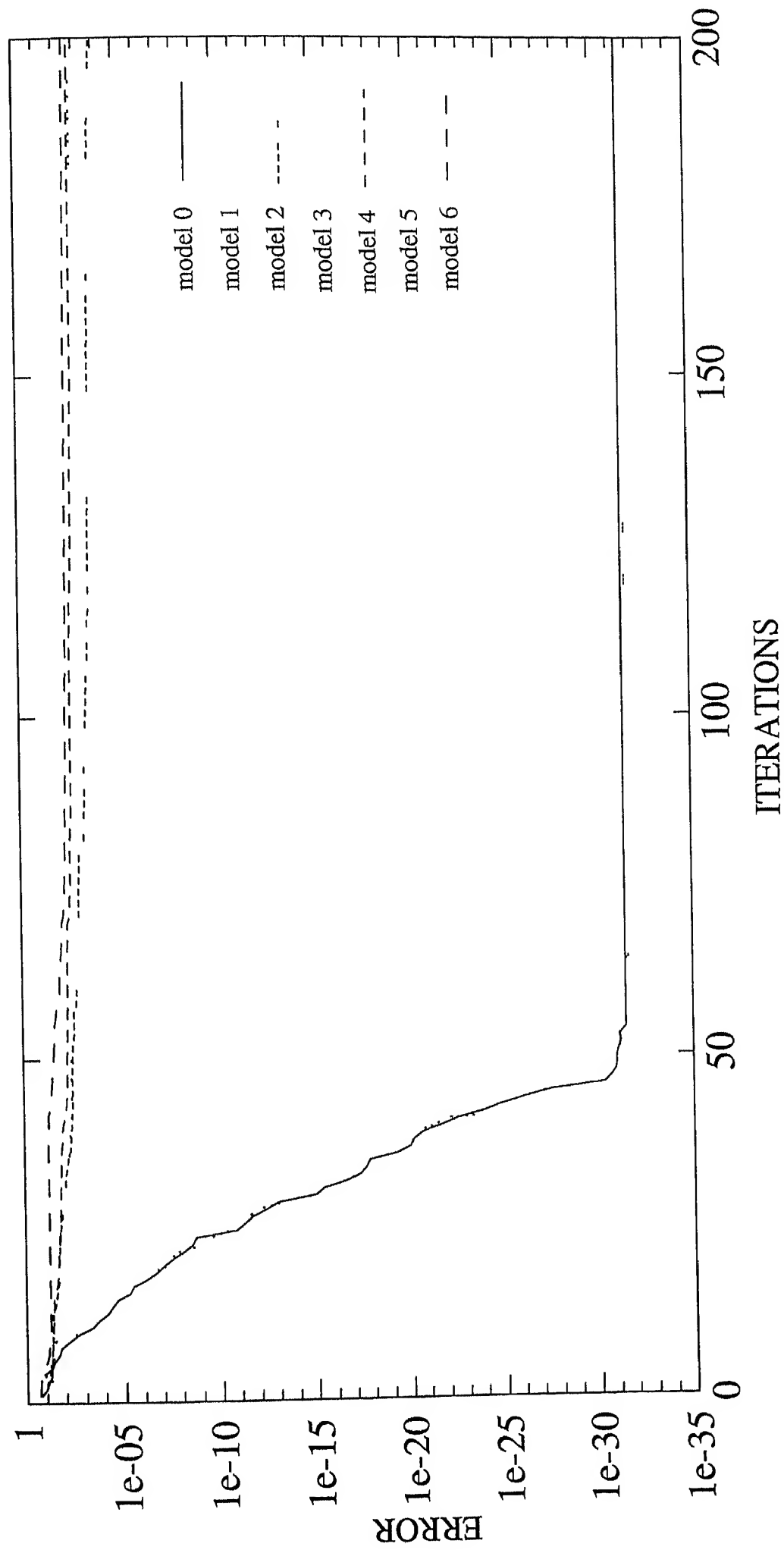


FIG 5 22 Error plot of Half adder truth problem for CNNA with 2 neurons

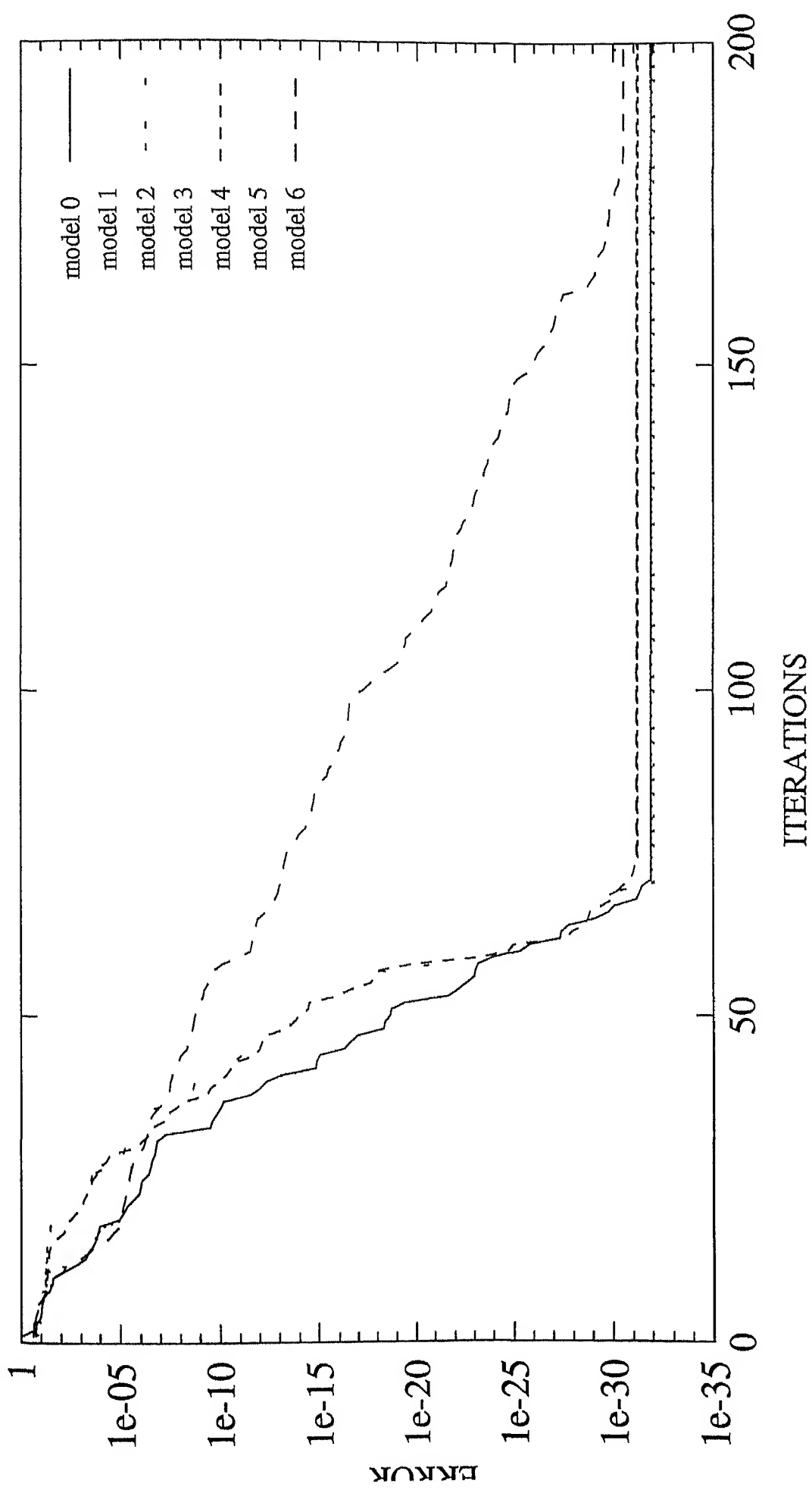


FIG 5 23 Error plot of Half adder truth problem for CNNA with 3 neurons

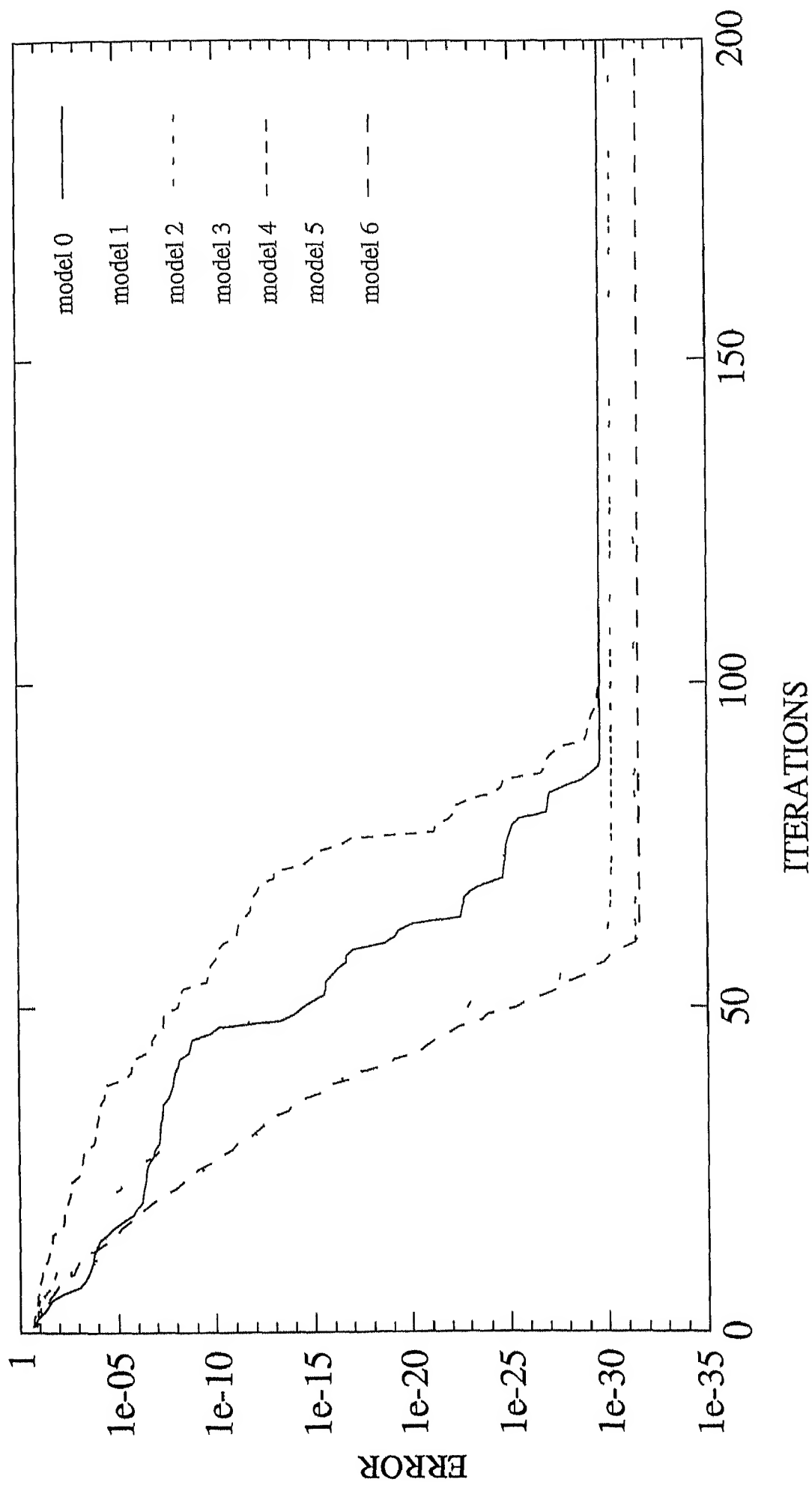


FIG 5 24 Error plot of Half adder truth problem for CNNA with 4 neurons

References

- [1] Di D K Chaturvedi P S Satsangi Prof P K Kalra Flexible Neural Network Models for Electrical Machines Semi Annual Paper Meeting IE(I) Journal vol 80 May 1999 pp 13-17
- [2] Vitell J F & Reifman I Premature saturation in backpropagation Networks Mechanism and necessary conditions *Neural Networks* Vol 10 No 4 1997 pp 721-735
- [3] Rumelhart D E McClelland J L and the PDP Research Group (eds) (1986) Parallel distributed processing: Exploration in micro structure of Cognition Vol I Foundations MIT press/Basilford Books Cambridge MA
- [4] Fahlman S E Fast learning variations on backpropagation: An empirical study in *Proc Connectionist Models Summer School* D S Touretzky G Hinton I Sejnowski 1988 pp 38-51
- [5] Mollet M L 'A scaled conjugate gradient algorithm for fast supervised learning' *Neural Networks* Vol 6 1993 pp 525-533
- [6] Johansson I M Dowling I U & Goodman D M Backpropagation learning for multi layer feedforward neural networks using the conjugate gradient method *Int J of Neural Systems* Vol 2 No 4 1991 pp 291-302
- [7] Watrous R & Shastri L Learning phonetic features using connectionist networks: an experiment in speech recognition in *Proc 1st IEEE Int Conf Neural Networks* June 1987
- [8] Moody J & Darken C 'Fast learning in network of locally tuned processing units *Neural Computation* Vol 1 1989 pp 281-294
- [9] Hopfield J 'Neural networks and physical systems with emergent collective computational abilities *Proc National academy of Science* 19 1982 pp 2554-2558
- [10] Zurada J M, "introduction to Artificial neural Systems

133616



133616

Date Slip

The book is to be returned on
the date last stamped

| |
|--|
| |
|--|



A133616